BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 620 522 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
09.05.2001 Bulletin 2001/19

(51) Int Cl.[7]: **G06F 9/445**

(21) Application number: **94301785.5**

(22) Date of filing: **14.03.1994**

(54) **High performance dynamic linking through caching**

Dynamische Hochleistungsprogrammverknüpfung durch Cachespeicherung

Liaison dynamique de haute performance par antémémorisation

(84) Designated Contracting States:
DE FR IT NL SE

(30) Priority: **13.04.1993 US 46827**

(43) Date of publication of application:
19.10.1994 Bulletin 1994/42

(73) Proprietor: **SUN MICROSYSTEMS, INC.**
Mountain View, California 94043-1100 (US)

(72) Inventors:
- **Nelson, Michael N.**
  San Carlos, California 94070 (US)
- **Hamilton, Graham**
  Palo Alto, California 94303 (US)

(74) Representative: **Wombwell, Francis**
Potts, Kerr & Co.
15, Hamilton Square
Birkenhead Merseyside L41 6BR (GB)

(56) References cited:
- **SOFTWARE PRACTICE & EXPERIENCE, vol. 21, no. 4, 1 April 1991 pages 375-390, XP 000147180 WILSON HO W ET AL 'AN APPROACH TO GENUINE DYNAMIC LINKING'**
- **IBM JOURNAL OF RESEARCH AND DEVELOPMENT, vol. 34, no. 1, 1 January 1990 pages 98-103, XP 000128185 AUSLANDER M A 'MANAGING PROGRAMS AND LIBRARIES IN AIX VERSION 3 FOR RISC SYSTEM/6000 PROCESSORS'**
- **PATENT ABSTRACTS OF JAPAN vol. 016 no. 495 (P-1436) ,14 October 1992 & JP-A-04 178731 (HITACHI LTD) 25 June 1992,**

EP 0 620 522 B1

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND OF THE INVENTION

Field of the Invention

[0001] The present invention relates to the fields of distributed computing systems, client-server computing and object oriented programming.

BACKGROUND

[0002] A computer programmer writes a program, the source program, in a high level language which typically makes use of symbols for addresses, and special characters or acronyms for operation codes. This source code comprises a set of instructions for the computer and data with which or upon which the instructions are to operate. These instructions and data must be loaded into a computer's memory at certain addresses in order for the computer to execute the program process. In order to make this happen, the source code is processed by a compiler which generates binary object code which a computer can execute. Before the computer can execute this newly written program, the program must go through several additional steps, during which the addresses in the program may be represented in different ways. The compiler typically will bind the symbolic addresses of the source code to relocatable addresses (such as 16 bytes from the beginning address of the program). A *linkage editor* or *loader* will in turn bind these relocatable addresses to absolute addresses (such as memory location 64216). Each binding is a mapping from one address space to another. This *binding* of instructions and data to memory addresses can be done either at compilation time, load time or at program execution time.

[0003] Referring to **Figure 1**, the several steps described above are depicted. The source program **1** is processed by a compiler **2** producing an object code module **3**. This object code module **3**, generally along with other previously compiled object modules **4**, is processed by a linkage editor **5** to produce a load module **6**. The load module **6** and any system libraries **7** required are processed by a loader **8** producing an in-memory binary image **10** of the original program and its related modules and libraries. This in-memory binary image **10** can now be executed by the computer.

[0004] Continuing to refer to **Figure 1**, when it is known at compile time **12** where the program will reside in memory, these programs are compiled with absolute code for addresses. In most cases however, it is not known at compile time where the program will reside in memory, and the compiler must generate relocatable code for the memory addresses. In this case, final binding of memory locations to the addresses is delayed until either load time **13** or execute time **14**. Some modern systems delay such address binding until execute time **14** when the program image can be moved during its execution from one memory segment to another or where the program start-up cost is not excessive because the program images contain few relocatable addresses, such as with position independent code (PIC). PIC is code generated by some compilers which can be placed anywhere in memory because all memory references are made relative to the *program counter*.

[0005] Modern computer operating systems are designed to optimize the use of memory space and to minimize user wait time. This is done in the address binding/program loading process, by *dynamic loading* of program object modules only when they are actually called by another module and *dynamic linking* of an object module to its system library routines only when they are required. In these cases the main program is loaded into memory and executed and supporting object modules or system libraries are not loaded unless they are called by the main program, thus saving memory space and load time at the expense of some program start-up time. Also, system libraries which will likely already be resident in memory can be dynamically linked to the executing main program when called, thereby not requiring a copy of the system libraries to be linked and loaded with each main program at load time, again saving memory space but at the cost of some program start-up time and some execution time. With dynamic linking, a *stub* is included in the image for each library-routine reference. This stub is a small piece of code that indicates how to locate the appropriate memory-resident library routine. When the stub is executed it replaces itself with the address of the routine and executes the routine. Under this scheme all programs that use a library routine use the same copy of the library code.

[0006] In order to take maximum advantage of dynamic linking and loading, program compilers must be designed to produce the necessary relocatable address references, the sub-routine stub code, and efficient PIC code. Unfortunately, existing compilers for object-oriented program modules cannot generate such code efficiently. For example, the cfront 3.0 preprocessor and the G++ compiler generate virtual function tables as initialized data structures which are full of references to relocatable symbols. Thus the number of relocatable symbols in object-oriented program modules is much higher than in more traditional program modules and the program startup delay required to dynamically link these modules can rise to unacceptable levels due to the number of relocations. Therefore what is required is a system that provides efficient dynamic linking of program modules with large numbers of relocatable symbols.

[0007] Additionally, IBM Journal of Research and Development, Volume 34, No. 1, 1 January 1990, Pages 98-103, discloses a program library matter in facility *in* an article entitled "Managing Programs and Library's in AIX Version 3 For RISC Sysem/6000 Processors" by M. A. Auslander.

[0008] The present invention as defined in the appended claims provides an elegant solution to this problem by caching linked program images and also caching partially linked library programs.

SUMMARY OF THE INVENTION

[0009] The present invention fills this need for minimizing the system delay caused by linking new object oriented programs by establishing an elegant and efficient system of caching fully bound program images along with their system library modules, and establishing a second level of caches for caching relocatable system library programs whose addresses are provisionally fixed-up (partly linked), and using these caches to supply fully linked or partially linked modules when a call to another program/routine is made. Thus if a new application is to be linked, the system first checks the image cache to see if the program has been linked before and if so the linked version of the program along with its linked library programs is used without further processing overhead. If the new application is not found in the image cache, then an attempt is made to minimize the overhead by seeing if some or all of the related library programs have already been provisionally fixed-up (partly linked), by checking a library program cache. Only if a library program is not found in the library program cache, does the linker system need to do a complete program load and link operation, thus minimizing the linking overhead.

[0010] A method is disclosed for a method to dynamically link a new program image and related library programs into an executable application program image. The method provides for producing a linked list of the required programs by giving the linker an argument representing the designated program image and a naming context which contains data on the associated library programs which are to be linked together. The linker finds all of the required programs, and links them together. The parent maps the program images into the designated addresses thereby completing the linking of the executable application program. In finding the required programs, the linker first checks the image cache to see if the new program and its related library programs is already linked and cached because it was executed before. If the new program is not found in the image cache, the linker object then checks the library program cache to see if the library programs are cached in partially linked form, and will use as many of these as it can find. For any library programs that must still be located, the linker will retrieve them from a data store and will then proceed to link all library programs and the new program image together to form an executable whole. This process of caching new program images with their library programs and caching partially linked library programs individually guarantees a procedure which minimizes the time delay in program start-up when a new program is executed.

DESCRIPTION OF THE DRAWINGS

[0011] The objects, features and advantages of the system of the present invention will be apparent from the following description in which:

Figure 1 illustrates the typical steps in compiling and loading a new program (Prior Art).
Figure 2 illustrates the SPRING operating system concept of an Object.
Figure 3 illustrates a High level flow chart of the invention.
Figure 4 illustrates a detailed flow chart of the invention.
Figure 5 illustrates a non-fixed-up program.
Figure 6 illustrates a fixed-up version of the program shown in **Figure 5.**
Figure 7 illustrates an example of a SPRING dynamic linking environment.

NOTATIONS AND NOMENCLATURE

[0012] The detailed descriptions which follow may be presented in terms of program procedures executed on a computer or network of computers. These procedural descriptions and representations are the means used by those skilled in the art to most effectively convey the substance of their work to others skilled in the art.

[0013] A procedure is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be noted, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

[0014] Further, the manipulations performed are often referred to in terms, such as adding or comparing, which are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or similar devices.

[0015] The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The procedures presented here-

in are not inherently related to a particular computer or other apparatus. Various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given.

DESCRIPTION OF THE PREFERRED EMBODIMENT

[0016] In the following description, for purposes of explanation, specific data and configurations are set forth in order to provide a thorough understanding of the present invention. The preferred embodiment described herein is implemented as a portion of the SPRING Object-Oriented Operating System created by Sun Microsystems®, Inc. (Sun Microsystems is a registered trademark of Sun Microsystems, Inc.) However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details and may be implemented in various computer systems and in various configurations, or makes or models of tightly-coupled processors or in various configurations of loosely-coupled multiprocessor systems. Moreover, it will be clear to those skilled in these arts that the present invention may be implemented in a non-object oriented computing system.

[0017] A SPRING *domain* is an *address space* with a collection of *threads*. A given domain may act as the server of some objects and the clients of other objects. The implementor or object manager and the client can be in the same domain or in a different domain.

The spring object model

[0018] SPRING has a slightly different way of viewing objects from other distributed object oriented systems and it is necessary to clarify this before discussing the details of the present invention.

[0019] Most distributed systems present a model wherein objects reside at server machines and client machines possess object handles that point to the object at the server. (See figure **2a**.) Clients pass around object handles rather than objects.

[0020] SPRING presents a model wherein clients are operating directly on objects, not on object handles. (See figure **2b**.) Some of these objects happen to keep all their interesting state at some remote site, so that their local state merely consists of a handle to this remote state. An object can only exist in one place at a time, so if an object is transmitted to someone else then the transmitter of the object ceases to have the object. However, the object can be copied before being transmitted, which might be implemented such that there are now two distinct objects pointing to the same remote state.

[0021] So whereas in systems such as MACH, one might talk of several clients having object handles that reference some remote object, in SPRING one would talk about several clients having objects that reference the same remote state.

Dynamic Linking of Process Images in SPRING

[0022] Referring now to **Figure 3** the basic steps of the present invention are briefly described. When a client domain in SPRING wishes to start a program, the client domain executes a "Link (Image Memory Object, Naming Context Object)" command **20** on a linker object. The linker object first checks his image cache **22** by looking-up the image memory object to see if it contains a fully-linked version of the program represented by the image memory object and that this cached version was linked using the library programs in the naming context object. If a match is found **28** the linker object is basically finished with its work and merely returns the cached list of memory objects and related addresses **38** which are associated with the found image memory object. If the check of the image cache produces no match **24** then the linker object finds a list of names of the library programs required by the image memory object and using the naming context object, obtains a memory object associated with each library program, and proceeds to check the library program cache for a match for each library program **26**. If a match is found for a library program memory object **36** the linker object obtains the memory object representing the provisionally fixed-up copy of the found library program and the associated address at which this program is to be mapped. If no match is found **30**, the linker object decides on an address at which to map the library program in question, creates a memory object that is a copy-on-write copy of the library program in question and maps this memory object at the address it selected for it. The linker object further provisionally fixes-up this new copy of the program and stores the library program's memory object and related address in the library program cache **32**. After finding a memory object for each library program required, whether they were found in the library program cache or new copies created, the linker object now resolves all unresolved symbol references in the new program image and in all of these library programs and writes a copy of this fully-linked version into the image cache **34**. Lastly the linker object returns to the client domain a list of the memory objects and related addresses for this fully-linked new program image and its associated library programs.

[0023] The above description is a summarized description of the present invention which is explained in more detail in the flow chart depicted in **Figure 4** and described in examples of the process below. Those skilled in the art will recognize that the use of names such an linker object, client domain etc. are for illustrative purposes and that the invention may be practiced by programs which may be called any kind of name.

[0024] Before proceeding to explain the invention in more detail, it is useful to describe an example of a "non-fixed-up program" and what happens as the relocations occur to make it a "fixed-up" or executable program image. Referring now to **Figur 5** a non-fixed-up program is depicted. File F **40** contains a program image. The file has four parts: a header **42** that contains the address **44** to begin executing the program and the list of shared library programs **46** to use; the code for the program **52**; a symbol table **48** that contains all external symbols defined in the program; and a relocation table **50** that contains a list of what symbols to relocate. In this example, the program header **42** says that the program needs shared library SL **46**. The relocation table **50** shows that the program code **52** contains an instruction at address A1 that references an external symbol *foo* **54.** In addition, the symbol table **48** shows that the program defines the external function *bar* **56** at address A2. Also shown in **Figure 5** is file F2 containing shared library SL **60**. File F2 has the same four parts as File F: a program header **62**; the code for the library program **72**; a symbol table **68**; and a relocation table **70**. Since this shared library does not depend on any other shared libraries, the shared library list is empty **66**. The relocation table **70** shows that the library code contains an instruction at address B I that references the external symbol *bar* **64**. In addition the symbol table **68** shows that the library code defines the external function *foo* at address B2 **76. Figure 6** shows a "fixed-up" version of the same program and library program which is depicted in **Figure 5. Figure 6** shows an address space **80** containing the fixed-up program code **82** and the fixed-up library code **84.** The code for the program image in File F (**52** in **Figure 5**) is mapped at the address **86** that is fixed for all program images. The value of this address is such that each code address in the image (**52** in **Figure 5**) is correct (for example, the code at address A1 (**58** in **Figure 5**) is mapped at address A1 in the address space **86**). The shared library SL on the other hand is mapped beginning at address space location address R1 and thus each library code address is offset by R**190** in the address space **80.** The external reference to the symbol *foo* in the instruction A1, which was shown in the File F relocation table (**54** in **Figure 5**) to be at instruction A1, was satisfied by the shared library. The symbol *foo* was defined at address B2 in the shared library (**76** in **Figure 5**) and since address B2 was mapped to address R1+B2 in the Address space **80,** the actual address put in A1 for *foo* was R1+B2 **88.** The external reference to the symbol *bar in* the instruction **B1** shown by the library relocation table (**64** in **Figure 5**) was satisfied by the program code which defined *bar* at address A2 in the program image **(56** in **Figure 5).** The instruction at address B1 in the shared library is actually mapped to address R1+B1 I in the address space **90** and so the actual address placed in the instruction at R1+B1 is A2 **92.**

[0025] Referring now to **Figure 7** an environment in an exemplary SPRING object oriented system is depicted with which to illustrate the preferred embodiment of the invention. Shown are: a Naming Service **100** containing an implementor for context object C **102**; a Domain D **104** containing an Address Space **106**; a Client Domain **108** containing a mapping list **120** and objects C **109**, L **110**, M **112,** AS **114**, D **116,** and DM **118;** a SPRING kernel **122,** containing a Domain Manager **124** which itself contains an implementor of domain object D **126** which is shown containing object AS **128**; a Virtual Memory Manager **130** also contained in the SPRING kernel which contains an address space implementor for objects AS **132**; a File Server Domain **152** containing implementors for memory object M **154**, memory object M2 **156**, memory object SL **158**, memory object SL2 **160** and memory object SL3 **162**; and finally a Linker Domain **140** containing an implementor for linker object L **142**, an image cache **144** containing an exemplary cache entry **146** which itself contains a list of memory objects and related addresses **172** and Memory objects C **164**, M **166**, M2 **168** and SL3 **170**, and a library cache **148** containing an exemplary cache entry **150** which itself contains memory objects SL **172** and SL2 **174**, and the cached program starting address **176.**

[0026] It should be noted that the following objects are equivalent: M **112** and M **166;** M2 **175** and M2 **168**; SL3 **177** and SL3 **170;** and C **109** and C **164.** Two objects for the purpose of this invention are considered equivalent if they reference the source state on the server.

[0027] In the examples which follow, a client domain wishes to start a program named *foo*. The configuration after the program is started is shown in **Figure 7.**

[0028] The simplest example is where there is a cache hit in the image cache. In this case starting the new domain goes through the following steps:

- The client domain **108** looks up foo via the Spring naming service **100** and gets back a memory object M **112** that stores the program image for foo. The program image stored in M needs shared library SL in order to be dynamically linked; the names of the libraries that an image needs to be linked are stored inside memory object M **112** whose implementation is on disk **154.**
- The client domain **108** looks up a linker object L **110** implemented by a linker domain **140** using the naming service **100.**
- The client domain **108** invokes the link method on L **110** passing in two parameters: memory object M **112** and a context object C **109** implemented by the naming service **100.**
- The linker domain **140** looks in its cache using M **112** and C **109** as its key and discovers that it has an entry with key M **166** and C **164** that matches (M **166** is equivalent to M **112** and C **164** is equivalent to C **109**). The cache entry contains a cached copy of a fully-linked version of M **166** that was linked using C **164.** The fully-linked version consists of two memory objects: M2 **168** and SL3 **170.** M2 **168** is a

fully linked copy of M 166 linked at address AI (the address to map program images is fixed for all programs to be the value AI) and SL3 **170** is a fully linked copy of the memory object for the shared library SL **172** linked at address A2 (the list of memory objects and corresponding linked addresses is shown at **173**).

- The linker domain **140** returns a list of <address to map, memory object pairs> **173** to the client domain **108** (this list is labeled mapping list **120** in the client domain **108**). The first entry in the list **121** contains address A1 and memory object M2. The second entry in the list **123** contains address A2, and memory object SL3.
- The client domain **108** looks up a domain manager object DM **118** using the naming service **100**.
- The client domain **108** invokes the *create_domain* method on object DM **118** and gets in return a new domain object D **116**.
- The client domain **108** invokes the *get_address_space* method on D **116** and gets an object AS **114** that represents D 's address space **132**.
- The client domain **108** invokes the *copy_and_map* method on the AS object **114** to map object M2 **175** at address A1 and SL3 **177** at address A2 copy-on-write. (Note that this information was obtained from the mapping list **120**).
- The client domain **108** starts domain D **126** running by invoking the *enter* method on domain object D **116**.

[0029] Note that in **Figure 7**, the library cache **148** also has a cached copy of shared library SL 172 and a provisionally fixed-up version which is in memory object SL2 **174**. However, since we got a hit on the image cache **144** we didn't need to consult the library cache **148**.

[0030] A second example is when there is no image cache hit but there is a library cache hit. For this example assume that the image cache **144** in **Figure 7** is empty when the client domain **108** tries to link the program image. The linker domain **140** does the link using the following steps:

- The linker domain **140** looks up M **112** and C **109** in its image cache **144** and does not find a match.
- The linker domain **140** looks in memory object M **112** and discovers that it needs a shared library program named SL to be linked.
- The linker domain **140** looks up the name SL using context object **C 109.** (Note that this step would be repeated for each library program required). The result is the memory object SL **300** (this memory object is not shown in the linker domain **140**) that contains the contents of the shared library program SL.
- The linker domain **140** looks in its library cache **148** using memory object SL **300** as an argument,and discovers that it has an entry with key SL **172** that watches (i.e., SL **172** is equivalent to SL **300**). The cached entry contains a cache copy of a provisionally fixed-up version of shared library SL which is memory object SL2 **174**.
- The linker domain **140** creates a memory object M2 **168** that is a copy-on-write copy of M **112** and creates a memory object SL3 **170** that is a copy-on-write copy of SL2 **174**.
- The linker domain **140** resolves all unresolved references in M2 **168** using SL3 **170** and resolves all unresolved references in SL3 **170** using M2 **168** (how this is done was described above and is depicted in the flow chart). The result is that M2 **168** and SL3 **170** comprise a fully-linked version of program *foo*. Again, it should be noted that if there were multiple library programs required, the same steps would be performed for each one.
- The linker domain enters M2 **168** and SL3 **170** into its image cache **144** along with their related keys (memory object C **164** and memory object M **166**) and returns a list of <address to map, memory object pairs> **173** to the client domain **108** (this list is labeled mapping list **120** in **108**). The first entry in the list **121** contains address AI and memory object M2 and the second entry in the list **123** contains address A2 and memory object SL3.

[0031] The client domain **108** then follows the same steps in the first example to create the new domain.

[0032] The last example is when there is no image or library cache hit. For this example assume that when the client domain **108** tries to link the program image, the image cache **144** and the library cache **148** are empty. The linker domain **140** does the link using the following steps:

- The linker domain **140** looks up M **112** and C **109** in its image cache **144** and does not find a match.
- The linker domain **140** finds memory object M **112** and maps it into its address space, and discovers that M **112** needs a shared library named SL to be linked.
- The linker domain **140** looks up the name **SL** using context object **C 109.** The result is the memory object SL **172** that contains the contents of the shared library program SL.
- The linker domain **140** looks in its library cache **148** using memory object SL **172** as the argument, and does not find a match (recall that in this example, we assumed the library cache to be empty).
- The linker domain **140** decides that memory object **SL 172** must be fixed-up at an available address in its address space (call it A2).
- The linker domain **140** creates a memory object **SL2 160** that is a copy-on-write copy of SL **172** and maps SL2 into its address space.
- The linker domain **140** provisionally fixes-up SL2

(see the discussion on fix-ups above and the detailed flow-chart in **Figure 4** for details) and stores SL2 **174** in the library cache **148** (along with its key SL **172**)..

- The linker domain **140** creates a memory object **M2 168** that is a copy-on-write copy of M **112** and creates a memory object SL3 **170** that is a copy -on-write copy of **SL2 174**.
- The linker domain **140** resolves all unresolved references in **M2 168** using SL3 **170** and resolves all unresolved references in SL3 **170** using M2 **168** (how this is done is explained in the discussion on fix-ups above and in the detailed flow chart in **Figure 4**). The result is that M2 **168** and SL3 **170** comprise a fully-linked version of program foo.
- The linker domain **140** enters **M2 168** and SL3 **170** into its image cache **144** (along with its key, memory object M**166** and its context object C **164**) and returns a list **172** of <address to map, memory object pairs> to the client domain **108**. The first entry of the list **121** contains address A1 and memory object M2. The second entry in the list **123** contains address A2 and memory object SL3.

[0033] The client domain **108** then follows the same steps as above in the first example, to create the new domain **104**.

[0034] An alternative embodiment of the invention is to use only the shared library program cache in the dynamic linker system, following the same steps as outlined above starting with the second example.

[0035] While the above examples describe the presently preferred embodiment, and describe specific program domains and a certain sequence of steps, those skilled in these arts will recognize that these specifics are not essential to the practice of the invention which basically embodies the use of caches for retaining fully linked images of programs and their associated libraries as well as partially fixed-up library routines with which to minimize the overall dynamic linking overhead cost.

Claims

1. A method of efficiently generating a fully-linked program image of a designated program and related library programs using a dynamic linking system having an image cache, the method implemented in a computer system, the method characterized by the steps of:

   invoking said linking system with parameters representing an unlinked program image of said designated program;
   searching (22) for said fully-linked program image of said designated program and said related library programs in said image cache; and
   if said fully-linked program image is found (28) in said image cache, then the method further includes the steps of:

   retrieving (38) said fully-linked program image with a list of program identifiers and related addresses for each said related library program from said image cache; and
   mapping (38) said fully-linked program image of said designated program and each of said related library programs at said addresses from said list of program identifiers.

2. The method of claim 1 wherein if the fully-linked program image is not found in said image cache, the method further comprises the steps of:

   generating (26) an unlinked copy of said designated program;
   searching(26) for a provisionally fixed-up copy of said related library programs in a library cache of said linking system; and
   if said provisionally fixed-up library programs is found in said library cache, then the method further including the steps of:

   retrieving (34) said provisionally fixed-up library programs from said library cache;
   linking (34) said provisionally fixed-up library, programs with each other and with said unlinked designated program to create said fully linked program image of said designated program and related library programs with a list of program identifiers and related addresses comprising an identifier and related address for said designated program and for each of said related library programs; and
   mapping (38) said fully-linked program image of said designated program and each of said provisionally fixed-up library programs at said related addresses from said list of program identifiers.

3. The method of claim 2 wherein if said provisionally fixed-up library programs is not found in said library cache, then the method further comprises the following steps:

   selecting an address (32) to which to link related library programs;
   generating an unlinked copy (32) of said related library programs;
   provisionally fixing-up relocation symbols (32) in said copy of said related library programs;
   linking (32) said provisionally fixed-up library programs with each other and with said unlinked designated program to create said fully

linked program image of said designated program and related library programs with a list of program identifiers and related addresses comprising an identifier and related address for said designated program and for each of said related library programs;
mapping (32) said fully-linked program image of said designated program and each of said related library programs at said related addresses from said list of program identifiers; and
adding (34) said provisionally fixed-up library programs to said library cache.

4. The method of claim 3 wherein if said program image is not found in said image cache then the method further comprises the steps of:
adding (34) to said image cache said fully-linked program image with said provisionally fixed-up library programs and said list of program identifiers and related addresses for subsequent retrieval by said linking system.

5. The method of claim 4 wherein all programs are object oriented programs.

6. The method of claim 1 wherein said step of searching for said fully-linked program image in said image cache comprises the step of:
checking said image cache for a key which matches said designated program and said related library programs.

7. The method of claim 2 wherein said step of searching for a provisionally fixed-up copy of said related library programs includes the additional steps of:

creating (308) a first name list containing names of said library programs which are related to said designated program; and
resolving (310) each of said names of said library programs contained on said first list into a library program identifier and adding each of said library program identifiers to a second list.

8. The method of claim 7 wherein said step of retrieving said provisionally fixed-up library programs includes the additional steps of:

checking (316) said library cache for a match for each library program identifier in said second list;
obtaining (322) a matching library program from said library cache in the event that a matching library program identifier is found; and
adding (336) said matching library program identifier from said library cache to a third list.

9. The method of claim 8 comprising the additional steps of:

selecting an address to which to link related library programs identified in said second list in the event that a matching library program identifier is not found in said library cache;
generating (330) an unlinked copy of said related library programs; provisionally (332) fixing-up relocation symbols in said copy of said related library programs;
linking (332) said provisionally fixed-up library programs with each other and with said unlinked designated program to create said fully linked program image of said designated program and related library programs with a list of program identifiers and related addresses comprising an identifier and related address for said designated program and for each of said related library programs;
adding (334) said provisionally fixed-up library programs to said library cache; and
adding (336) said provisionally fixed-up library program identifier to said third list.

10. The method of claim 9 comprising the additional steps of:

making a copy (342) of said program image and adding an identifier of said copy of said program image to a fourth list, said copy of said program image having a symbol table and a relocation table; and
making a copy (344) of each of said library programs from said third list and adding an identifier of said copy of said library programs to said fourth list, each of said copies of said library programs have a symbol table and a relocation table.

11. The method of claim 10 comprising the additional steps of:

finding each symbol (352) to be relocated in said copy of said program image; and
searching (336) all of said library programs identified in said fourth list and in the event that a first symbol definition which matches said symbol to be relocated is found, using an address of said first symbol definition which matches said symbol to be relocated, to perform a relocation for said symbol to be relocated.

12. The method of claim 11 comprising the additional steps of:

using said symbol table (356) in each of said

library programs identified in said fourth list in turn, starting with a first of said library programs identified in said fourth list, comparing each symbol of said symbol table to said program image's symbol table and marking a symbol in said library program's symbol table as overridden in the event that said symbol matches a symbol in said program image's symbol table;
comparing each symbol of each of said symbol tables to all of said symbol tables of said library programs identified in said fourth list which are higher in said fourth list than said symbol table being checked and which have been marked as already checked;
upon completion of the check of said symbol table of each of said library program identified in said fourth list, marking said library program's identifier in said fourth list as one whose symbol table has been checked; and
comparing said symbols in said symbol tables in each of said library programs identified in said fourth list in like manner until all of said symbol tables have been compared.

13. The method of claim 12 comprising the additional steps of:

using said relocation table (366) in each of said programs identified in said fourth list in turn, starting with a first of said programs identified in said fourth list, scanning each of said relocation tables to find an undefined symbol;
in the event that a symbol in one of said relocation tables is a defined symbol (370), checking said defined symbol and in the event said defined symbol has been marked as overridden, undoing a relocation address for said defined symbol which has been marked as overridden;
using each undefined symbol (368) and each defined symbol whose relocation address was undone, as a search argument, checking said symbol table of said copy of said program image and said symbol tables of each of said library programs identified in said fourth list to find a symbol matching said search argument, and in the event said matching symbol is found, using said found symbol's address to perform said address relocation for said symbol which was used as said search argument; and
providing an exception message (372) in the event no symbol is found to match said search argument.

14. The method of claim 13 comprising the additional steps of:

creating a cachable list (388) of program identifiers, containing a copy of said designated program image and each of said library programs identified in said fourth list;
entering said cachable list (388) of program identifiers into said image cache; and
returning said cachable list (390) of program identifiers in response to said linking system.

15. The method of claim 14 wherein said programs are objects and said computing system is an object oriented system.

16. The method of claim 15 wherein said identifiers and said program identifiers are memory objects.

17. The method of claim 1, wherein:

if said fully-linked program image is not found (24) in said image cache, then the method further including the steps of:

checking a library cache (26) for a provisionally fixed-up copy of said related library programs and in the event that said library cache contains said provisionally fixed-up related library programs retrieving said provisionally fixed up library programs; and if said provisionally fixed up library programs is not found in said library cache (30), then the method further including the steps of:

creating (32) a list of library program identifiers and related addresses from said library cache;
linking (34) each of said library programs to each other and to said program image to create said fully-linked program image; and
mapping (38) said fully-linked program image identified by said program identifiers at said related address from said list of program identifiers.

18. The method of claim 17 wherein said step of creating a list of library program identifiers and related addresses from said library program cache comprises the additional steps of:

creating (308) a first list containing names of said library programs which are related to said designated program;
resolving (310) each of said names of said library programs contained on said first list into a library program identifier and adding each of said library program identifiers to a second list;
checking (322) said library cache for a match for each library program identifier in said sec-

ond list;
obtaining (336) a matching library program from said library cache in the event that a matching library program identifier is found; and
adding (336) said matching library program identifier from said library cache to a third list.

19. The method of claim 18 comprising the additional steps of:

selecting (328) an address to which to link related library programs identified in said second list in the event that a matching library program identifier is not found in said library cache;
generating (330) an unlinked copy of said related library programs;
provisionally (332) fixing-up relocation symbols in said copy of said related library programs;
adding (334) said provisionally fixed-up library programs to said library cache; and
adding (336) said provisionally fixed-up library program identifier to said third list.

20. The method of claim 19 comprising the additional steps of:

making (342) a copy of said program image and adding an identifier of said copy of said program image to a fourth list, said copy of said program image having a symbol table and a relocation table; and
making (342) a copy of each of said library programs from said third list and adding an identifier of said copy of said library programs to said fourth list, each of said copies of said library programs having a symbol table and a relocation table.

21. The method of claim 20 comprising the additional steps of:

finding (352) each symbol to be relocated in said copy of said program image; and
searching (356) all of said library programs identified in said fourth list and in the event that a first symbol definition which matches said symbol to be relocated is found, using an address of said first symbol definition which matches said symbol to be relocated, to perform a relocation for said symbol to be relocated.

22. The method of claim 21 comprising the additional steps of:

using (358) said symbol table in each of said library programs identified in said fourth list in turn, starting with a first of said library programs identified in said fourth list, comparing each symbol of said symbol table to said program image's symbol table and marking a symbol in said library program's symbol table as overridden in the event that said symbol matches a symbol in said program image's symbol table;
comparing each symbol of each of said symbol tables to all of said symbol tables of said library programs identified in said fourth list which are higher in said fourth list than said symbol table being checked and which have been marked as already checked;
upon completion of the check of said symbol table of each of said library program identified in said fourth list, marking said library program's identifier in said fourth list as one whose symbol table has been checked; and
comparing said symbols in said symbol tables in each of said library programs identified in said fourth list in like manner until all of said symbol tables have been compared.

23. The method of claim 22 comprising the additional steps of:

using (366) said relocation table in each of said programs identified in said fourth list in turn, starting with a first of said programs identified in said fourth list, scanning each of said relocation tables to find an undefined symbol;
in the event that a symbol in one of said relocation tables is a defined symbol, checking said defined symbol (378) a second time and in the event said defined symbol has been marked as overridden, undoing a relocation address for said defined symbol which has been marked as overridden;
using each undefined symbol (368) and each defined symbol whose relocation address was undone as a search argument, checking said symbol table of said copy of said program image and said symbol tables of each of said library programs identified in said fourth list to find a symbol matching said search argument, and in the event said matching symbol is found, using said found symbol's address to perform said address relocation for said symbol which was used as said search argument; and
providing an exception message (372) in the event no symbol is found to match said search argument.

24. The method of claim 23 comprising the additional steps of:

creating (388) a cachable list of program identifiers, containing a copy of said designated

program image and each of said library programs identified in said fourth list; and
returning (390) said cachable list of program identifiers in response to said linking system.

25. The method of claim 24 wherein said programs are objects and said computing system is an object oriented system.

26. The method of claim 25 wherein said identifiers and said program identifiers are memory objects.

27. An image caching system for providing a fully-linked program image of a designated program and related library programs to a dynamic linking system, the caching system being implemented in a computer system, the image caching system characterized by:

an image cache (144) for caching said fully-linked program image with a list of program identifiers and related addresses for each said related library program, said list for mapping said fully-linked program image;
a first searching facility (22), coupled to said image cache, for searching said image cache to determine whether said image cache contains said fully-linked program image, said search being made in response to a request to link said designated program; and
a first communications facility (38), coupled to said first searching facility, for responding to said request to link said designated program, said response containing said list of program identifiers and addresses related to said program identifiers obtained from said image cache in the event that said fully linked-program image is found in said image cache.

28. The image caching system as recited in claim 27 further comprising:

a library cache (148), coupled to said image cache, for caching provisionally fixed-up library programs; and
a second searching facility (263), coupled to said library cache, for searching said library cache to determine whether at least one of said provisionally fixed-up library programs matches at least one of said related library programs.

29. The image caching system as recited in claim 28 wherein said search of said library cache is made in the event that said fully-linked program image is not found in said image cache which matches said designated program, and wherein said first searching facility obtains an unlinked copy of said designated program.

30. The image caching system as recited in claim 29 further comprising:

a first linking facility (32), coupled to said second searching facility, for linking a found library program to said unlinked program image, in the event that at least one of said provisionally fixed-up library programs is found in said library cache which matches at least one of said related library programs.

31. The image caching system as recited in claim 30 further comprising:

a third searching facility (36), coupled to said second searching facility, for finding unlinked library programs in the event that a library program is not found in said library cache which matches said related library programs; and
a second linking facility (32), coupled to said third searching facility, for linking said unlinked library programs to each other and to said found library programs and to said unlinked program image, producing fully-linked program image of said designated program with related library programs.

32. The image caching system as recited in claim 31 wherein:

said second linking facility (32) provisionally fixes-up any of said unlinked library programs and stores said provisionally fixed-up library programs in said library cache; and
said second linking facility (32) stores said fully-linked program image with related library programs in said image cache (144).

33. The image caching system as recited in claim 32 wherein said designed program and library programs are object oriented programs.

34. The image caching system as recited in claim 28, wherein said second searching facility (26) performing said search in response to a request to link said designated program when said fully-linked program image of said designated program is not found by the dynamic linking system; and

a first linking facility (38) coupled to said second searching facility, for linking a found library program to said designated program, in the event that said found library program is found in said library cache.

35. The image caching system as recited in claim 34 wherein said first searching facility further comprises:

a second searching facility (26), coupled to said library cache (148), for searching said library

cache to determine whether said library cache contains an unlinked library program which matches one of said related library program.

36. The image caching system as recited in claim 34 further comprising:

a second searching facility (26), coupled to said memory of said computer, for finding said unlinked library programs which matches said related library programs; and
a second linking facility (32), coupled to said second searching facility, for linking said unlinked library programs to each other and to said found library programs and to said unlinked copy of said designated program, producing a fully-linked program image of said designated program with related library programs.

37. The image caching system as recited in claim 36 wherein:
said second linking facility (32) provisionally fixes-up any of said unlinked library programs and stores said provisionally fixed-up library programs in said library cache.

38. The Dynamic Linking system as recited in claim 37 wherein said library programs in said library program cache (148) are partially linked library programs.

39. The Dynamic Linking system as recited in claim 38 wherein said application programs and said library programs are object oriented programs.

Patentansprüche

1. Ein Verfahren zum effektiven Erzeugen eines vollständig verknüpften Programmabbilds eines bestimmten Programms und zugehöriger Bibliotheksprogramme unter Verwendung eines dynamischen Verknüpfungssystems, das einen Abbild-Cache umfaßt, wobei das Verfahren in einem Computersystem implementiert ist, wobei das Verfahren gekennzeichnet ist durch die Schritte:

Aufrufen des Verknüpfungssystems mit Parametern, die ein unverknüpftes Programmabbild des bestimmten Programms darstellen;
Durchsuchen (22) des Abbild-Caches nach dem vollständig verknüpften Programmabbild des bestimmten Programms und der zugehörigen Bibliotheksprogramme; und
wobei dann, wenn das vollständig verknüpfte Programmabbild in dem Abbild-Cache gefunden wird (28), das Verfahren ferner die Schritte umfaßt:

Wiedergewinnen (38) des vollständig verknüpften Programmabbilds mit einer Liste von Programmidentifizierern und zugehörigen Adressen für jedes der zugehörigen Bibliotheksprogramme aus dem Abbild-Cache; und
Abbilden (38) des vollständig verknüpften Programmabbilds des bestimmten Programms und jedes der zugehörigen Bibliotheksprogramme auf die Adressen aus der Liste von Programmidentifizierern.

2. Das Verfahren nach Anspruch 1, wobei dann, wenn das vollständig verknüpfte Programmabbild nicht in dem Abbild-Cache gefunden wird, das Verfahren ferner die Schritte umfaßt:

Erzeugen (26) einer unverknüpften Kopie des bestimmten Programms;
Durchsuchen (26) eines Bibliotheks-Cache des Verknüpfungssystems nach einer provisorisch festgelegten (fixed-up) Kopie der zugehörigen Bibliotheksprogramme; und
wobei dann, wenn die provisorisch festgelegten Bibliotheksprogramme in dem Bibliotheks-Cache gefunden werden, das Verfahren ferner die Schritte umfaßt:

Wiedergewinnen (34) der provisorisch festgelegten Bibliotheksprogramme aus dem Bibliotheks-Cache;
Verknüpfen (34) der provisorisch festgelegten Bibliotheksprogramme miteinander und mit dem unverknüpften bestimmten Programm, um das vollständig verknüpfte Programmabbild des bestimmten Programms und der zugehörigen Bibliotheksprogramme mit einer Liste von Programmidentifizierern und zugehörigen Adressen, die einen Identifizierer und eine zugehörige Adresse für das bestimmte Programm und für jedes der zugehörigen Bibliotheksprogramme enthält, zu erzeugen; und
Abbilden (38) des vollständig verknüpften Programmabbilds des bestimmten Programms und jedes der provisorisch festgelegten Bibliotheksprogramme auf die zugehörigen Adressen aus der Liste von Programmidentifizierern.

3. Das Verfahren nach Anspruch 2, wobei dann, wenn die provisorisch festgelegten Bibliotheksprogramme nicht in dem Bibliotheks-Cache gefunden werden, das Verfahren ferner die folgenden Schritte umfaßt:

Auswählen einer Adresse (32), mit welcher zugehörige Bibliotheksprogramme verknüpft werden sollen;
Erzeugen einer unverknüpften Kopie (32) der zugehörigen Bibliotheksprogramme;
provisorisches Festlegen (fixing-up) von Verschiebungssymbolen (32) in der Kopie der zugehörigen Bibliotheksprogramme;
Verknüpfen (32) der provisorisch festgelegten Bibliotheksprogramme miteinander und mit dem unverknüpften bestimmten Programm, um das vollständig verknüpfte Programmabbild des bestimmten Programms und der zugehörigen Bibliotheksprogramme mit einer Liste von Programmidentifizierern und zugehörigen Adressen, die einen Identifizierer und eine zugehörige Adresse für das bestimmte Programm und für jedes der zugehörigen Bibliotheksprogramme umfaßt, zu erzeugen;
Abbilden (32) des vollständig verknüpften Programmabbilds des bestimmten Programms und jedes der zugehörigen Bibliotheksprogramme auf die zugehörigen Adressen aus der Liste von Programmidentifizierern; und
Hinzufügen (34) der provisorisch festgelegten Bibliotheksprogramme zu dem Bibliotheks-Cache.

4. Das Verfahren nach Anspruch 3, wobei dann, wenn das Programmabbild nicht in dem Abbild-Cache gefunden wird, das Verfahren ferner die Schritte umfaßt:

Hinzufügen (34) des vollständig verknüpften Programmabbilds mit den provisorisch festgelegten Bibliotheksprogrammen und der Liste von Programmidentifizierern und zugehörigen Adressen zu dem Abbild-Cache für eine nachfolgende Wiedergewinnung durch das Verknüpfungssystem.

5. Das Verfahren nach Anspruch 4, wobei sämtliche Programme objekt-orientierte Programme sind.

6. Das Verfahren nach Anspruch 1, wobei der Schritt des Durchsuchens des Abbild-Cache nach dem vollständig verknüpften Programmabbild den Schritt umfaßt:

Überprüfen des Abbild-Cache nach einem Schlüssel, welcher mit dem bestimmten Programm und den zugehörigen Bibliotheksprogrammen übereinstimmt.

7. Das Verfahren nach Anspruch 2, wobei der Schritt des Durchsuchens nach einer provisorisch festgelegten Kopie der zugehörigen Bibliotheksprogramme die zusätzlichen Schritte umfaßt:

Erzeugen (308) einer ersten Namensliste, die Namen der Bibliotheksprogramme enthält, welche zu dem bestimmten Programm gehören; und
Auflösen (310) jedes der Namen der in der ersten Liste enthaltenen Bibliotheksprogramme in einen Bibliotheksprogrammidentifizierer und Hinzufügen jedes Bibliotheksprogrammidentifizierers zu einer zweiten Liste.

8. Das Verfahren nach Anspruch 7, wobei der Schritt des Wiedergewinnens der provisorisch festgelegten Bibliotheksprogramme die zusätzlichen Schritte umfaßt:

Überprüfen (316) des Bibliotheks-Cache hinsichtlich einer Übereinstimmung jedes Bibliotheksprogrammidentifizierers in der zweiten Liste;
Gewinnen (322) eines übereinstimmenden Bibliotheksprogramms aus dem Bibliotheks-Cache, sofern ein übereinstimmender Bibliotheksprogrammidentifizierer gefunden wird; und
Hinzufügen (336) des übereinstimmenden Bibliotheksprogrammidentifizierers aus dem Bibliotheks-Cache zu einer dritten Liste.

9. Das Verfahren nach Anspruch 8, umfassend die zusätzlichen Schritte:

Auswählen einer Adresse, mit welcher in der zweiten Liste identifizierte zugehörige Bibliotheksprogramme verknüpft werden sollen, sofern kein übereinstimmender Bibliotheksprogrammidentifizierer in dem Bibliotheks-Cache gefunden wird;
Erzeugen (330) einer unverknüpften Kopie der zugehörigen Bibliotheksprogramme;
provisorisches Festlegen (332) von Verschiebungssymbolen in der Kopie der zugehörigen Bibliotheksprogramme;
Verknüpfen (332) der provisorisch festgelegten Bibliotheksprogramme miteinander und mit dem unverknüpften bestimmten Programm, um das vollständig verknüpfte Programmabbild des bestimmten Programms und der zugehörigen Bibliotheksprogramme mit einer Liste von Programmidentifizierern und zugehörigen Adressen, die einen Identifizierer und eine zugehörige Adresse für das bestimmte Programm und für jedes der zugehörigen Bibliotheksprogramme enthält, zu erzeugen;
Hinzufügen (334) der provisorisch festgelegten Bibliotheksprogramme zu dem Bibliotheks-Cache; und
Hinzufügen (336) der provisorisch festgelegten Bibliotheksprogrammidentifizierer zu der dritten Liste.

10. Das Verfahren nach Anspruch 9, umfassend die zu-

sätzlichen Schritte:

Herstellen einer Kopie (342) des Programmabbilds und Hinzufügen eines Identifizierers der Kopie des Programmabbilds zu einer vierten Liste, wobei die Kopie des Programmabbilds eine Symboltabelle und eine Verschiebungstabelle aufweist; und Herstellen einer Kopie (344) jedes der Bibliotheksprogramme aus der dritten Liste und Hinzufügen eines Identifizierers der Kopie der Bibliotheksprogramme zu der vierten Liste, wobei jede der Kopien der Bibliotheksprogramme eine Symboltabelle und eine Verschiebungstabelle aufweist.

11. Das Verfahren nach Anspruch 10, umfassend die zusätzlichen Schritte:

Auffinden jedes zu verschiebenden Symbols (352) in der Kopie des Programmabbilds; und Durchsuchen (336) sämtlicher der in der vierten Liste identifizierten Bibliotheksprogramme und, sofern eine erste Symboldefinition, welche mit dem zu verschiebenden Symbol übereinstimmt, gefunden wird, Verwenden einer Adresse der ersten Symboldefinition, welche mit dem zu verschiebenden Symbol übereinstimmt, um eine Verschiebung für das zu verschiebende Symbol durchzuführen.

12. Das Verfahren nach Anspruch 11, umfassend die zusätzlichen Schritte:

Verwenden der Symboltabelle (356) in jedem der in der vierten Liste identifizierten Bibliotheksprogramme, um der Reihe nach, beginnend mit einem ersten in der vierten Liste identifizierten Bibliotheksprogramm, jedes Symbol der Symboltabelle mit der Symboltabelle des Programmabbilds zu vergleichen und um ein Symbol in der Symboltabelle des Bibliotheksprogramms als überschrieben (overridden) zu markieren, sofern das Symbol mit einem Symbol in der Symboltabelle des Programmabbilds übereinstimmt;
Vergleichen jedes Symbols jeder der Symboltabellen mit sämtlichen Symboltabellen der in der vierten Liste identifizierten Bibliotheksprogramme, welche an einer höheren Stelle in der vierten Liste sind, als die Symboltabelle, welche überprüft wird, und welche als bereits überprüft markiert worden sind;
Markieren des Identifizierers eines in der vierten Liste identifizierten Bibliotheksprogramms bei Abschluß der Überprüfung der Symboltabelle des Bibliotheksprogramms der vierten Liste als einen, dessen Symboltabelle überprüft worden ist; und
Vergleichen der Symbole in den Symboltabellen in jedem der in der vierten Liste identifizierten Bibliotheksprogramme auf eine gleiche Weise, bis sämtliche Symboltabellen verglichen worden sind.

13. Das Verfahren nach Anspruch 12, umfassend die zusätzlichen Schritte:

Verwenden der Verschiebungstabelle (366) in jedem der in der vierten Liste identifizierten Programme, um nacheinander beginnend mit einem ersten in der vierten Liste identifizierten Programm jede der Verschiebungstabellen zu durchsuchen, um ein undefiniertes Symbol aufzufinden;
sofern ein Symbol in einer der Verschiebungstabellen ein definiertes Symbol ist (370), Überprüfen des definierten Symbols, und sofern das definierte Symbol als überschrieben markiert worden ist, Rückgängigmachen einer Verschiebungsadresse für das definierte Symbol, welches als überschrieben markiert worden ist;
Verwenden jedes undefinierten Symbols (368) und jedes definierten Symbols, dessen Verschiebungsadresse rückgängig gemacht wurde, als Suchargument beim Überprüfen der Symboltabelle der Kopie des Programmabbilds und der Symboltabellen jeder der in der vierten Liste identifizierten Bibliotheksprogramme, um ein mit dem Suchargument übereinstimmendes Symbol aufzufinden, und, sofern das übereinstimmende Symbol gefunden ist, Verwenden der Adresse des übereinstimmenden Symbols, um die Adreßverschiebung für das Symbol durchzuführen, welches als Suchargument verwendet worden ist; und
Bereitstellen einer Ausnahmenachricht (372) für den Fall, daß herausgefunden wird, daß kein Symbol mit dem Suchargument übereinstimmt.

14. Das Verfahren nach Anspruch 13, umfassend die zusätzlichen Schritte:

Erzeugen einer cache-baren Liste (388) von Programmidentifizierern, die eine Kopie des bestimmten Programmabbilds und jedes der in der vierten Liste identifizierten Bibliotheksprogramme enthält;
Eingeben der cache-baren Liste (388) von Programmidentifizierern in den Abbild-Cache; und
Rückgeben der cache-baren Liste (390) von Programmidentifizierern in Erwiderung auf das Verknüpfungssystem.

15. Das Verfahren nach Anspruch 14, wobei die Programme Objekte sind und das Computersystem ein objekt-orientiertes System ist.

16. Das Verfahren nach Anspruch 15, wobei die Identifizierer und die Programmidentifizierer Speicherobjekte sind.

17. Das Verfahren nach Anspruch 1, wobei:

dann, wenn kein vollständig verknüpftes Programmabbild in dem Abbild-Cache gefunden wird (24), das Verfahren ferner die Schritte umfaßt:

Überprüfen eines Bibliotheks-Cache (26) hinsichtlich einer provisorisch festgelegten (fixed-up) Kopie der zugehörigen Bibliotheksprogramme und dann, wenn der Bibliotheks-Cache die provisorisch festgelegten zugehörigen Bibliotheksprogramme enthält, Wiedergewinnen der provisorisch festgelegten Bibliotheksprogramme; und
wobei dann, wenn die provisorisch festgelegten Bibliotheksprogramme in dem Bibliotheks-Cache nicht gefunden werden (30), das Verfahren ferner die Schritte umfaßt:

Erzeugen (32) einer Liste von Bibliotheksprogrammidentifizierern und zugehörigen Adressen aus dem Bibliotheks-Cache;
Verknüpfen (34) jedes der Bibliotheksprogramme miteinander und mit dem Programmabbild, um das vollständig verknüpfte Programmabbild zu erzeugen; und
Abbilden (38) des von den Programmidentifizieren identifizierten vollständig verknüpften Programmabbilds mit der zugehörigen Adresse aus der Liste von Programmidentifizierern.

18. Das Verfahren nach Anspruch 17, wobei der Schritt des Erzeugens einer Liste von Bibliotheksprogrammidentifizierern und zugehörigen Adressen aus dem Bibliotheksprogramm-Cache die zusätzlichen Schritte umfaßt:

Erzeugen (308) einer ersten Liste, die die Namen derjenigen Bibliotheksprogramme enthält, welche zu dem bestimmten Programm gehören;
Auflösen (310) jedes der Namen der Bibliotheksprogramme, die in der ersten Liste enthalten sind, in einen Bibliotheksprogrammidentifizierer und Hinzufügen jedes der Bibliotheksprogrammidentifizierer zu einer zweiten Liste;
Überprüfen (322) des Bibliotheks-Cache hinsichtlich einer Übereinstimmung für jeden Bibliotheksprogrammidentifizierer in der zweiten Liste;
Gewinnen (336) eines übereinstimmenden Bibliotheksprogramms aus dem Bibliotheks-Cache, sofern ein übereinstimmender Bibliotheksprogrammidentifizierer gefunden wird; und
Hinzufügen (336) des übereinstimmenden Bibliotheksprogrammidentifizierers aus dem Bibliotheks-Cache zu einer dritten Liste.

19. Das Verfahren nach Anspruch 18, umfassend die zusätzlichen Schritte:

Auswählen (328) einer Adresse, mit welcher in der zweiten Liste identifizierte zugehörige Bibliotheksprogramme verknüpft werden sollen, sofern kein übereinstimmender Bibliotheksprogrammidentifizierer in dem Bibliotheks-Cache gefunden wird;
Erzeugen (330) einer unverknüpften Kopie der zugehörigen Bibliotheksprogramme;
provisorisches (332) Festlegen von Verschiebungssymbolen in der Kopie der zugehörigen Bibliotheksprogramme;
Hinzufügen (334) der provisorisch festgelegten Bibliotheksprogramme zu dem Bibliotheks-Cache; und
Hinzufügen (336) des provisorisch festgelegten Bibliotheksprogrammidentifizierers zu der dritten Liste.

20. Das Verfahren nach Anspruch 19, umfassend die zusätzlichen Schritte:

Herstellen (343) einer Kopie des Programmabbilds und Hinzufügen eines Identifizierers der Kopie des Programmabbilds zu einer vierten Liste, wobei die Kopie des Programmabbilds eine Symboltabelle und eine Verschiebungstabelle aufweist; und
Herstellen (342) einer Kopie jedes der Bibliotheksprogramme aus der dritten Liste und Hinzufügen eines Identifizierers der Kopie der Bibliotheksprogramme zu der vierten Liste, wobei jede der Kopien der Bibliotheksprogramme eine Symboltabelle und eine Verschiebungstabelle aufweist.

21. Das Verfahren nach Anspruch 20, umfassend die zusätzlichen Schritte:

Auffinden (352) jedes zu verschiebenden Symbols in der Kopie des Programmabbilds; und
Durchsuchen (356) sämtlicher in der vierten Liste identifizierter Bibliotheksprogramme und, sofern eine erste Symboldefinition, welche mit dem zu verschiebenden Symbol übereinstimmt, aufgefunden wird, Verwenden einer Adresse der ersten Symboldefinition, welche

mit dem zu verschiebenden Symbol übereinstimmt, um eine Verschiebung des zu verschiebenden Symbols durchzuführen.

22. Das Verfahren nach Anspruch 21, umfassend die zusätzlichen Schritte:

Verwenden (358) der Symboltabelle in jedem der in der vierten Liste identifizierten Bibliotheksprogramme, um der Reihe nach beginnend mit einem ersten der in der vierten Liste identifizierten Bibliotheksprogramme jedes Symbol der Symboltabelle mit der Symboltabelle des Programmabbilds zu vergleichen und ein Symbol in der Symboltabelle des Bibliotheksprogramms als überschrieben zu markieren, sofern das Symbol mit einem Symbol in der Symboltabelle des Programmabbilds übereinstimmt;
Vergleichen jedes Symbols jeder der Symboltabellen mit sämtlichen Symboltabellen der in der vierten Liste identifizierten Bibliotheksprogramme, welche an einer höheren Stelle in der vierten Liste sind als die Symboltabelle, welche überprüft wird und welche als bereits überprüft markiert worden ist;
bei Abschluß der Überprüfung der Symboltabelle jedes der in der vierten Liste identifizierten Bibliotheksprogramme: Markieren des Identifizierers des Bibliotheksprogramms in der vierten Liste als einen, dessen Symboltabelle überprüft worden ist; und
Vergleichen der Symbole in den Symboltabellen in jedem der in der vierten Liste identifizierten Bibliotheksprogramme in gleicher Weise, bis sämtliche Symboltabellen verglichen worden sind.

23. Das Verfahren nach Anspruch 22, umfassend die zusätzlichen Schritte:

Verwenden (366) der Verschiebungstabelle in jedem der in der vierten Liste identifizierten Programme, um nacheinander beginnend mit einem ersten der in der vierten Liste identifizierten Programme jede der Verschiebungstabellen zu durchsuchen, um ein undefiniertes Symbol aufzufinden;
dann, wenn ein Symbol in einer der Verschiebungstabellen ein definiertes Symbol ist, Überprüfen des definierten Symbols (378) ein zweites Mal und, sofern das definierte Symbol als überschrieben markiert worden ist, Rückgängig-Machen einer Verschiebungsadresse für das definierte Symbol, welches als überschrieben markiert worden ist;
Verwenden jedes undefinierten Symbols (368) und jedes definierten Symbols, dessen Verschiebungsadresse rückgängig gemacht wurde, als Suchargument, Überprüfen der Symboltabelle der Kopie des Programmabbilds und der Symboltabellen jeder der in der vierten Liste identifizierten Bibliotheksprogramme, um ein Symbol aufzufinden, das mit dem Suchargument übereinstimmt, und, sofern das übereinstimmende Symbol gefunden wird, Verwenden der Adresse des gefundenen Symbols, um die Adreßverschiebung für das Symbol, welches als das Suchargument verwendet worden ist, durchzuführen; und
Bereitstellen einer Ausnahmenachricht (372), sofern gefunden wird, daß kein Symbol mit dem Suchargument übereinstimmt.

24. Das Verfahren nach Anspruch 23, umfassend die zusätzlichen Schritte:

Erzeugen (388) einer cache-baren Liste von Programmidentifizierern, die eine Kopie des bestimmten Programmabbilds und jedes der in der vierten Liste identifizierten Bibliotheksprogramme enthält; und
Zurückgeben (390) der cache-baren Liste von Programmidentifizierern in Erwiderung auf das Verknüpfungssystem.

25. Das Verfahren nach Anspruch 24, wobei die Programme Objekte sind und das Computersystem ein objekt-orientiertes System ist.

26. Das Verfahren nach Anspruch 25, wobei die Identifizierer und die Programmidentifizierer Speicherobjekte sind.

27. Ein Abbild-Cache-System zum Bereitstellen eines vollständig verknüpften Programmabbilds eines bestimmten Programms und zugehöriger Bibliotheksprogramme für ein dynamisches Verknüpfungssystem, wobei das Cache-System in einem Computersystem implementiert ist, wobei das Abbild-Cache-System gekennzeichnet ist durch:

einen Abbild-Cache (144) zum Cache-Speichern des vollständig verknüpften Programmabbilds mit einer Liste von Programmidentifizierern und zugehörigen Adressen für jedes der zugehörigen Bibliotheksprogramme, wobei die Liste zum Abbilden des vollständig verknüpften Programmabbilds dient;
eine erste mit dem Abbild-Cache gekoppelte Sucheinrichtung (22) zum Durchsuchen des Abbild-Cache, um zu bestimmen, ob der Abbild-Cache das vollständig verknüpfte Programmabbild enthält, wobei die Suche in Beantwortung einer Anforderung zum Verknüpfen des bestimmten Programms durchgeführt wird;

und
eine mit der ersten Sucheinrichtung gekoppelte erste Kommunikationseinrichtung (38) zum Antworten auf die Anforderung zum Verknüpfen des bestimmten Programms, wobei die Antwort die Liste von Programmidentifizierern und zu den Programmidentifizierern gehörende Adressen enthält, die aus dem Abbild-Cache gewonnen wurden, sofern das vollständig verknüpfte Programmabbild in dem Abbild-Cache gefunden wird.

28. Das Abbild-Cache-System nach Anspruch 27, ferner aufweisend:

einen mit dem Abbild-Cache gekoppelten Bibliotheks-Cache (148) zum Speichern provisorisch festgelegter (fixed-up) Bibliotheksprogramme; und
eine mit dem Bibliotheks-Cache gekoppelte zweite Sucheinrichtung (263) zum Durchsuchen des Bibliotheks-Cache, um zu bestimmen, ob wenigstens eines der provisorisch festgelegten Bibliotheksprogramme mit wenigstens einem der zugehörigen Bibliotheksprogramme übereinstimmt.

29. Das Abbild-Cache-System nach Anspruch 28, wobei das Durchsuchen des Bibliotheks-Cache durchgeführt wird, wenn das vollständig verknüpfte Programmabbild nicht in dem Abbild-Cache, welcher mit dem bestimmten Programm übereinstimmt, gefunden wird, und wobei die erste Suchmöglichkeit eine unverknüpfte Kopie des bestimmten Programms gewinnt.

30. Das Abbild-Cache-System nach Anspruch 29, ferner aufweisend:
eine mit der zweiten Sucheinrichtung gekoppelte erste Verknüpfungseinrichtung (32) zum Verknüpfen eines gefundenen Bibliotheksprogramms mit dem unverknüpften Programmabbild, sofern wenigstens eines der provisorisch festgelegten Bibliotheksprogramme in dem Bibliotheks-Cache gefunden ist, welches mit wenigstens einem der zugehörigen Bibliotheksprogramme übereinstimmt.

31. Das Abbild-Cache-System nach Anspruch 30, ferner aufweisend:

eine mit der zweiten Sucheinrichtung gekoppelte dritte Sucheinrichtung (36) zum Auffinden unverknüpfter Bibliotheksprogramme, sofern kein Bibliotheksprogramm in dem Bibliotheks-Cache gefunden wird, das mit dem zugehörigen Bibliotheksprogramm übereinstimmt; und
eine mit der dritten Sucheinrichtung gekoppelte zweite Verknüpfungseinrichtung (32) zum Verknüpfen der unverknüpften Bibliotheksprogramme miteinander und mit den gefundenen Bibliotheksprogrammen und mit dem unverknüpften Programmabbild, wobei ein vollständig verknüpftes Programmabbild des bestimmten Programms mit zugehörigen Bibliotheksprogrammen erzeugt wird.

32. Das Abbild-Cache-System nach Anspruch 31, wobei:

eine zweite Verknüpfungseinrichtung (32) irgendwelche der unverknüpften Bibliotheksprogramme provisorisch festlegt (fixed-up) und die provisorisch festgelegten Bibliotheksprogramme in dem Bibliotheks-Cache speichert; und
die zweite Verknüpfungseinrichtung (32) das vollständig verknüpfte Programmabbild mit zugehörigen Bibliotheksprogrammen in dem Abbild-Cache (144) speichert.

33. Das Abbild-Cache-System nach Anspruch 22, wobei das bestimmte Programm und die Bibliotheksprogramme objekt-orientierte Programme sind.

34. Das Abbild-Cache-System nach Anspruch 28, wobei die zweite Sucheinrichtung (26) die Suche in Erwiderung einer Anforderung zum Verknüpfen des bestimmten Programms durchführt, sofern das vollständig verknüpfte Programmabbild des bestimmten Programms von dem dynamischen Verknüpfungssystem nicht aufgefunden wird; und
eine mit der zweiten Sucheinrichtung gekoppelte erste Verknüpfungseinrichtung (38) zum Verknüpfen eines aufgefundenen Bibliotheksprogramms mit dem bestimmten Programm, sofern das gefundene Bibliotheksprogramm in dem Bibliotheks-Cache gefunden wird.

35. Das Abbild-Cache-System nach Anspruch 34, wobei die erste Sucheinrichtung ferner aufweist:
eine mit dem Bibliotheks-Cache (148) gekoppelte zweite Sucheinrichtung (26) zum Durchsuchen des Bibliotheks-Cache, zum Bestimmen, ob der Bibliotheks-Cache ein unverknüpftes Bibliotheksprogramm enthält, das mit einem der zugehörigen Bibliotheksprogramme übereinstimmt.

36. Das Abbild-Cache-System nach Anspruch 34, ferner aufweisend:

eine mit dem Speicher des Computers gekoppelte zweite Sucheinrichtung (26) zum Auffinden der unverknüpften Bibliotheksprogramme, welche mit den zugehörigen Bibliotheksprogrammen übereinstimmen; und
eine mit der zweiten Sucheinrichtung gekoppelte zweite Verknüpfungseinrichtung (32) zum

Verknüpfen der unverknüpften Bibliotheksprogramme miteinander und mit den aufgefundenen Bibliotheksprogrammen und mit der unverknüpften Kopie des bestimmten Programms, wobei ein vollständig verknüpftes Programmabbild des bestimmten Programms mit zugehörigen Bibliotheksprogrammen erzeugt wird.

37. Das Abbild-Cache-System nach Anspruch 36, wobei:

die zweite Verknüpfungseinrichtung (32) irgendwelche der unverknüpften Bibliotheksprogramme provisorisch festlegt (fixed-up) und die provisorisch festgelegten Bibliotheksprogramme in den Bibliotheks-Cache speichert.

38. Das dynamische Verknüpfungssystem nach Anspruch 37, wobei die Bibliotheksprogramme in dem Bibliotheksprogramm-Cache (148) partiell verknüpfte Bibliotheksprogramme sind.

39. Das dynamische Verknüpfungssystem nach Anspruch 38, wobei die Anwendungsprogramme und die Bibliotheksprogramme objekt-orientierte Programme sind.

Revendications

1. Procédé pour la génération efficace d'une image de programme entièrement liée d'un programme désigné et de programmes de librairie attenants à l'aide d'un système de liaison dynamique possédant une antémémoire d'image, procédé mise en oeuvre dans un système informatique, procédé caractérisé par les étapes suivantes :

   - l'appel dudit système de liaison avec des paramètres représentant une image de programme non liée dudit programme désigné ;
   - la recherche (22) de ladite image de programme entièrement lié dudit programme désigné et desdits programmes de librairie attenants dans ladite antémémoire d'image ; et
   - si ladite image de programme entièrement liée est trouvée (28) dans ladite antémémoire d'image, le procédé comprend alors, de plus, les étapes suivantes :
     - l'extraction (38) de ladite image de programme entièrement liée avec une liste d'identificateurs de programmes et d'adresses attenantes pour chaque dit programme de librairie attenant à partir de ladite antémémoire d'image ; et
     - le mappage (38) de ladite image de programme entièrement liée dudit programme désigné et de chacun desdits programmes de librairie attenants sur lesdites adresses à partir de ladite liste d'identificateurs de programme.

2. Procédé selon la revendication 1, selon lequel, si l'image de programme entièrement liée n'est pas trouvée dans ladite antémémoire d'image, le procédé comprend, de plus, les étapes suivantes :

   - la génération (26) d'une copie non liée dudit programme désigné ;
   - la recherche (26) d'une copie agencée, de façon temporaire, desdits programmes de librairie attenants dans une antémémoire de librairie dudit système de liaison ; et
   - si lesdits programmes de librairie agencés, de façon temporaire, sont trouvés dans ladite antémémoire de librairie, le procédé comprend alors, de plus, les étapes suivantes :
     - l'extraction (34) desdits programmes de librairie agencés, de façon temporaire, à partir de ladite antémémoire de librairie ;
     - la liaison (34) desdits programmes de librairie agencés, de façon temporaire, l'un avec l'autre et avec ledit programme désigné non lié afin de créer ladite image de programme entièrement liée dudit programme désigné et des programmes de librairie attenants avec une liste d'identificateurs de programmes et d'adresses attenantes comprenant un identificateur et une adresse attenante pour ledit programme désigné et pour chacun desdits programmes de librairie attenant ; et
     - le mappage (38) de ladite image de programme entièrement liée dudit programme désigné et de chacun desdits programmes de librairie agencés, de façon temporaire, sur lesdites adresses attenantes à partir de ladite liste d'identificateurs de programme.

3. Procédé selon la revendication 2, selon lequel, si lesdits programmes de librairie agencés, de façon temporaire, ne sont pas trouvés dans ladite antémémoire de librairie, le procédé comprend alors, de plus, les étapes suivantes :

   - la sélection d'une adresse (32) à laquelle on effectuera une liaison desdits programmes de librairie attenantes ;

- la génération d'une copie non liée (32) desdits programmes de librairie attenants ;

- l'agencement, de façon temporaire, de symboles de relocalisation (32) dans ladite copie desdits programmes de librairie attenants ;

- la liaison (32) desdits programmes de librairie agencés, de façon temporaire, l'un avec l'autre et avec ledit programme désigné non lié afin de créer ladite image de programme entièrement liée dudit programme désigné et des programmes de librairie attenants avec une liste d'identificateurs de programmes et d'adresses attenantes comprenant un identificateur et des adresses attenantes pour ledit programme désigné et pour chacun desdits programmes de librairie attenants ;

- le mappage (32) de ladite image de programme entièrement liée dudit programme désigné et de chacun desdits programmes de librairie attenants sur lesdites adresses attenantes à partir de ladite liste d'identificateurs de programme ; et

- l'ajout (34) desdits programmes de librairie agencés, de façon temporaire, à ladite antémémoire de librairie.

4. Procédé selon la revendication 3, selon lequel, si ladite image de programme n'est pas trouvée dans ladite antémémoire d'image, le procédé comprend alors, de plus, l'étape suivante :

   - l'ajout (34) à ladite antémémoire d'image de ladite image de programme entièrement liée avec lesdits programmes de librairie agencés, de façon temporaire, et de ladite liste d'identificateurs de programme et d'adresses attenantes pour une extraction ultérieure par ledit système de liaison.

5. Procédé selon la revendication 4, selon lequel tous les programmes sont des programmes orientés objet.

6. Procédé selon la revendication 1, selon lequel ladite étape de recherche de ladite image de programme entièrement liée dans ladite antémémoire d'image comprend l'étape suivante :

   - la vérification de ladite antémémoire d'image en ce qui concerne une clé correspondant audit programme désigné et auxdits programmes de librairie attenants.

7. Procédé selon la revendication 2, selon lequel ladite étape de recherche d'une copie agencée, de façon temporaire, desdits programmes de librairie attenants comprend les étapes additionnelles suivantes :

   - la création (308) d'une première liste de noms contenant des noms desdits programmes de librairie qui sont relatifs audit programme désigné ; et

   - la résolution (310) de chacun desdits noms desdits programmes de librairie contenus dans ladite première liste en un identificateur de programme de librairie et l'ajout de chacun desdits identificateurs de programme de librairie à une seconde liste.

8. Procédé selon la revendication 7, selon lequel ladite étape d'extraction desdits programmes de librairie agencés, de façon temporaire, comprend les étapes additionnelles suivantes :

   - la vérification (316) de ladite antémémoire de librairie en ce qui concerne une correspondance pour chaque identificateur de programme de librairie dans ladite seconde liste ; et

   - l'obtention (322) d'un programme de librairie correspondant à partir de ladite antémémoire de librairie dans le cas où un identificateur correspondant de programme de librairie est trouvé ; et

   - l'ajout (336) dudit identificateur correspondant de programme de librairie à partir de ladite antémémoire de librairie à une troisième liste.

9. Procédé selon la revendication 8, comprenant les étapes additionnelles suivantes :

   - la sélection d'une adresse à laquelle on doit lier les programmes de librairie attenants identifiés dans ladite seconde liste dans le cas où un identificateur correspondant de programme de librairie n'est pas trouvé dans ladite antémémoire de librairie ;

   - la génération (330) d'une copie non liée desdits programmes de librairie attenants ;

   - l'agencement temporaire (332) de symboles de relocalisation dans ladite copie desdits programmes de librairie attenants ;

   - la liaison (332) desdits programmes de librairie agencés, de façon temporaire, l'un avec l'autre et avec ledit programme désigné non lié afin de créer ladite image de programme entièrement

liée dudit programme désigné et des programmes de librairie attenants avec une liste d'identificateurs de programmes et d'adresses attenantes comprenant un identificateur et des adresses attenantes pour ledit programme désigné et pour chacun desdits programmes de librairie attenants ;

- l'ajout (334) desdits programmes de librairie agencés, de façon temporaire, à ladite antémémoire de librairie ; et

- l'ajout (336) dudit identificateur de programme de librairie agencés, de façon temporaire, à ladite troisième liste.

10. Procédé selon la revendication 9, comprenant les étapes additionnelles suivantes :

- la réalisation d'une copie (342) de ladite image de programme et l'ajout d'un identificateur de ladite copie de ladite image de programme à une quatrième liste, ladite copie de ladite image de programme possédant une table de symbole et une table de relocalisation ; et

- la réalisation d'une copie (344) de chacun desdits programmes de librairie à partir de ladite troisième liste et l'ajout d'un identificateur de ladite copie desdits programmes de librairie à ladite quatrième liste, chacune desdites copies desdits programmes de librairie possédant une table de symboles et une table de relocalisation.

11. Procédé selon la revendication 10, comprenant les étapes additionnelles suivantes :

- trouver chaque symbole (352) à relocaliser dans la dite copie de ladite image de programme ;

- rechercher (336) tous lesdits programmes de librairie identifiés dans ladite quatrième liste et dans le cas où une première définition de symbole correspondant audit symbole à relocaliser est trouvée, utiliser une adresse de ladite première définition de symbole correspondant audit symbole à relocaliser, afin d'effectuer une relocalisation dudit symbole à relocaliser.

12. Procédé selon a revendication 11, comprenant les étapes additionnelles suivantes :

- l'utilisation de ladite table de symboles (356) dans chacun desdits programmes de librairie identifiés à son tour dans ladite quatrième liste, le démarrage par un premier desdits programmes de librairie identifié dans ladite quatrième liste, la comparaison de chaque symbole de ladite table de symboles avec ladite table de symboles d'image de programme et le marquage d'un symbole dans ladite table de symbole de programme de librairie comme étant supplanté dans le cas où ledit symbole correspond à un symbole dans ladite table de symboles d'image de programme ;

- la comparaison de chaque symbole de chacune desdites tables de symboles avec toutes lesdites tables de symboles desdits programmes de librairie identifiés dans ladite quatrième liste qui sont plus hauts dans ladite quatrième liste que ladite table de symboles vérifiée et qui ont été marqués comme étant déjà vérifiés ;

- à la fin de la vérification de ladite table de symboles de chacun desdits programmes de librairie identifiés dans ladite quatrième liste, le marquage dudit identificateur de programme de librairie dans ladite quatrième liste comme possédant une table de symboles vérifiée ; et

- la comparaison desdits symboles dans lesdites tables de symboles dans chacun desdits programmes de librairie identifiés dans ladite quatrième liste de façon similaire jusqu'à ce que toutes lesdites tables de symboles aient été vérifiées.

13. Procédé selon la revendication 12, comprenant les étapes additionnelles suivantes :

- l'utilisation de ladite table de relocalisation (366) à leur tour dans chacun desdits programmes de librairie identifiés dans ladite quatrième liste, le démarrage par un premier desdits programmes identifiés dans ladite quatrième liste, le balayage de chacune desdites tables de relocalisation afin de trouver un symbole non défini ;

- dans le cas où un symbole dans une desdites tables de relocalisation est un symbole défini (370) la vérification dudit symbole défini et dans le cas où ledit symbole défini a été marqué comme étant supplanté, la non-affectation d'une adresse de relocalisation pour ledit symbole défini qui a été marqué comme étant supplanté ;

- l'utilisation de chaque symbole non défini (368) et de chaque symbole défini dont l'adresse de relocalisation n'a pas été affectée, comme argument de recherche, la vérification de ladite table de symboles de ladite copie de ladite ima-

ge de programme et desdites tables de symboles desdits programmes de librairie identifiés dans ladite quatrième liste afin de trouver un symbole correspondant audit argument de recherche, t dans le cas où ledit symbole correspondant est trouvé, l'utilisation de ladite adresse de symbole trouvé pour effectuer ladite relocalisation d'adresse pour ledit symbole qui a été utilisé comme ledit argument de recherche ; et

- la fourniture d'un message d'exception (372) dans le cas où aucun symbole n'est trouvé comme correspondant audit argument de recherche.

14. Procédé selon la revendication 13, comprenant les étapes additionnelles suivantes :

- la création d'une liste pouvant être placée en antémémoire (388) d'identificateurs de programme, contenant une copie de ladite image désignée de programme et de chacun desdits programmes de librairie identifiés dans ladite quatrième liste ;

- l'entrée de ladite liste pouvant être placée en antémémoire (388) d'identificateurs de programme dans ladite antémémoire d'image ; et

- le renvoi de ladite liste pouvant être placée en antémémoire (390) d'identificateurs de programme en réponse audit système de liaison.

15. Procédé selon la revendication 14, selon lequel lesdits programmes sont des objets et ledit système informatique est un système orienté objet.

16. Procédé selon la revendication 15, selon lequel lesdits identificateurs et lesdits identificateurs de programme sont des objets de mémoire.

17. Procédé selon la revendication 1, selon lequel :
si ladite image de programme entièrement liée n'est pas trouvée (24) dans ladite antémémoire d'image, le procédé comprend, de plus, les étapes suivantes :

- la vérification d'une antémémoire de librairie (26) en ce qui concerne une copie agencée, de façon temporaire, desdits programmes attenants de librairie et dans le cas où l'antémémoire de librairie contient lesdits programmes de librairie attenants agencés, de façon temporaire, l'extraction desdits programmes de librairie agencés de façon temporaire ; et

- si lesdits programmes de librairie agencés de façon temporaire ne sont pas trouvés dans ladite antémémoire de librairie (30), le procédé comprends alors, de plus, les étapes suivantes :

  - la création (32) d'une liste d'identificateurs de librairie et d'adresses attenantes à partir de ladite antémémoire de librairie ;

  - la liaison (34) desdits programmes de librairie l'un avec l'autre et à ladite image de programme afin de créer ladite image de programme entièrement liée ; et

  - le mappage (38) de ladite image de programme entièrement liée identifiée par lesdits identificateurs de programme auxdites adresses attenantes à partir de ladite liste d'identificateurs de programme.

18. Procédé selon la revendication 17, selon lequel ladite étape de création d'une liste d'identificateurs de programme et d'adresses attenantes à partir de ladite antémémoire de programmes de librairie comprend les étapes additionnelles suivantes :

- la création (308) d'une première liste comprenant les noms desdits programmes de librairie qui sont liés auxdits programmes désignés ;

- la séparation (310) de chacun desdits noms de programmes de librairie contenus dans ladite première liste dans un identificateur de programme de librairie et l'ajout de chacun desdits identificateurs de programme de librairie à une seconde liste ;

- la vérification (322) de ladite antémémoire de librairie en ce qui concerne une correspondance pour chaque identificateur de programme de librairie de ladite seconde liste ;

- l'obtention (336) d'un programme de libraire correspondant à partir de ladite antémémoire de librairie dans le cas où un identificateur de programme de librairie correspondant est trouvé ; et

- l'ajout (336) dudit identificateur de programme de librairie correspondant à partir de ladite antémémoire de librairie à une troisième liste.

19. Procédé selon la revendication 18, comprenant les étapes additionnelles suivantes :

- la sélection (328) d'une adresse à laquelle on doit lier les programmes de librairie attenants identifiés dans ladite seconde liste dans le cas

où un identificateur de programme de librairie correspondant n'est pas trouvé dans ladite antémémoire de librairie ;

- la génération (330) d'une copie non liée desdits programmes de libraire attenants ;
- l'agencement temporaire (332) de symboles de relocalisation dans ladite copie desdits programmes de librairie attenants ;
- l'ajout (334) desdits programmes de librairie agencés de façon temporaire à ladite antémémoire de librairie ; et
- l'ajout (336) dudit identificateur de programme de librairie agencé de façon temporaire à ladite troisième liste.

20. Procédé selon la revendication 19, comprenant les étapes additionnelles suivantes :

- la réalisation (332) d'une copie de ladite image de programme et l'ajout d'un identificateur de ladite copie de ladite image de programme à une quatrième liste, ladite copie de ladite image de programme possédant une table de symboles et une table de relocalisation ; et
- la réalisation (342) d'une copie de chacun desdits programmes de librairie à partir de ladite troisième liste et l'ajout d'un identificateur de ladite copie desdits programmes de librairie à ladite quatrième liste, chacune desdites copies desdits programme de librairie possédant une table de symboles et une table de relocalisation.

21. Procédé selon la revendication 20, comprenant les étapes additionnelles suivantes :

- la reconnaissance (352) de chaque symbole à relocaliser dans ladite copie de ladite image de programme ; et
- la recherche (356) de tous lesdits programmes de librairie identifiés dans ladite quatrième liste et dans le cas où une première définition de symbole correspondant audit symbole à relocaliser est trouvée, l'utilisation d'une adresse de ladite première définition de symbole correspondant audit symbole à relocaliser pour effectuer une relocalisation dudit symbole à relocaliser.

22. Procédé selon la revendication 21, comprenant les étapes additionnelles suivantes :

- l'utilisation (358) de ladite table de symboles dans chacun desdits programmes de librairie identifiés à leur tour dans ladite quatrième liste, en commençant par un premier desdits programmes de librairie identifiés dans ladite quatrième liste, la comparaison de chaque symbole de ladite table de symboles avec ladite table de symboles de l'image de programme et le marquage d'un symbole dans ladite table de symboles de programme de librairie comme étant supplanté dans le cas où ledit symbole correspond à un symbole de ladite table de symboles d'image de programme ;
- la comparaison de chaque symbole de ladite table de symboles avec toutes lesdites tables de symboles desdits programmes de librairie identifiés dans ladite quatrième liste qui sont plus hautes dans ladite quatrième liste que ladite table de symboles vérifiée et qui ont été marquées comme étant déjà vérifiées ;
- la fin de la vérification de ladite table de symboles de chacun desdits programmes de librairie identifiés dans ladite quatrième liste, le marquage dudit identificateur de programme de librairie de ladite quatrième liste comme ayant une table de symboles vérifiée ; et
- la comparaison desdits symboles desdites tables de symbole dans chacun desdits programmes de librairie identifiés dans ladite quatrième liste de façon similaire jusqu'à ce que toutes lesdites tables de symboles aient été comparées.

23. Procédé selon la revendication 22, comprenant les étapes additionnelles suivantes :

- l'utilisation (366) de ladite table de relocalisation dans chacun desdits programmes identifiés dans ladite quatrième liste à son tour, en commençant par un premier desdits programmes identifiés dans ladite quatrième liste, balayant chacune desdites tables de relocalisation pour trouver un symbole non défini ;
- dans le cas où un symboles d'une desdites tables de relocalisation est un symbole défini, la vérification dudit symbole défini (378) une seconde fois et dans le cas où ledit symbole défini a été marqué comme étant supplanté, l'annulation d'une adresse de relocalisation pour ledit symbole défini qui a été marqué comme étant supplanté ;
- l'utilisation de chaque symbole non-défini (368) et de chaque symbole défini dont l'adresse de

relocalisation a été annulée comme argument de recherche, la vérification de ladite table de symboles de ladite copie de ladite image de programme et desdites tables de symboles de chacun desdits programmes de librairie identifiés dans ladite quatrième liste pour trouver un symbole correspondant audit argument de recherche, et dans le cas où ledit symbole correspondant est trouvé, l'utilisation de ladite adresse de symbole trouvé pour effectuer ladite relocalisation d'adresse pour ledit symbole qui a été utilisé comme ledit argument de recherche ; et

- la fourniture d'une message d'exception (372) dans le cas où aucun symbole n'a été trouvé comme correspondant audit argument de recherche.

24. Procédé selon la revendication 23, comprenant les étapes additionnelles suivantes :

- la création (388) d'une liste pouvant être placée en antémémoire d'identificateurs de programme, contenant une copie de ladite image de programme désigné et de chacun desdits programmes de librairie identifiés dans ladite quatrième liste ; et

- le renvoi (390) de ladite liste pouvant être placée en antémémoire d'identificateurs de programme en réponse audit système de liaison.

25. Procédé selon la revendication 24, selon lequel lesdits programmes sont des objets et ledit système informatique est un système orienté objet.

26. Procédé selon la revendication 25, selon lequel lesdits identificateurs et lesdits identificateurs de programme sont des objets de mémoire.

27. Système de mise en antémémoire d'image pour fournir une image de programme entièrement liée d'un programme désigné et des programmes de librairie attenants à un système de liaison dynamique, le système de mise en antémémoire étant mis en oeuvre dans un système informatique, le système de mise en antémémoire d'image état caractérisé par :

- une antémémoire d'image (144) pour mettre en antémémoire ladite image de programme entièrement liée avec une liste d'identificateurs de programme et d'adresses attenantes pour chaque dit programme de librairie attenant, ladite liste étant prévue pour le mappage de ladite image de programme entièrement liée ;

- un premier moyen de recherche (22), couplé à ladite antémémoire d'image, pour rechercher dans ladite antémémoire d'image afin de déterminer si ladite antémémoire d'image contient ladite image de programme entièrement liée, ladite recherche étant effectuée en réponse à une demande de liaison dudit programme désigné ;

- un premier moyen de communication (38), couplé audit premier moyen de recherche, pour répondre à ladite demande de liaison dudit programme désigné, ladite réponse contenant ladite liste d'identificateurs de programme et d'adresses attenantes auxdits identificateurs de programme obtenus à partir de ladite antémémoire d'image dans le cas où ladite image de programme entièrement liée est trouvée dans ladite antémémoire d'image.

28. Système de mise en antémémoire d'image selon la revendication 27, comprenant, de plus :

- une antémémoire de librairie (148), couplée à ladite antémémoire d'image, pour la mise en antémémoire de programmes de librairie agencés de façon temporaire ; et

- un second moyen de recherche (263), couplé à ladite antémémoire de librairie, pour rechercher dans ladite antémémoire de librairie pour déterminer si au moins un desdits programmes de librairie agencés de façon temporaire correspond à au moins un desdits programmes de librairie attenants.

29. Système de mise en antémémoire d'image selon la revendication 28, dans lequel ladite recherche de ladite antémémoire de librairie est effectuée dans le cas où ladite image de programme entièrement liée n'est pas trouvée dans ladite antémémoire d'image correspondant audit programme désigné, et dans lequel ledit premier moyen de recherche obtient une copie non liée dudit programme désigné.

30. Système de mise en antémémoire d'image selon la revendication 29, comprenant, de plus :

- un premier moyen de liaison (32), couplé audit second moyen de recherche, pour lier un programme de librairie trouvé à ladite image de programme non liée, dans le cas où au moins un desdits programmes de librairie agencés de façon temporaire est trouvé dans ladite antémémoire de librairie correspondant à au moins un desdits programmes de librairie attenants.

31. Système de mise en antémémoire d'image selon la

revendication 30, comprenant, de plus :

- un troisième moyen de recherche (36), couplé audit second moyen de recherche, pour trouver des programmes de librairie non liés dans la cas où un programme de librairie n'est pas trouvé dans ladite antémémoire de librairie comme correspondant auxdits programmes de librairie attenants ; et

- un second moyen de liaison (32), couplé audit troisième moyen de recherche, pour lier lesdits programmes de librairie non liés l'un avec l'autre et auxdits programmes de librairie trouvés et à ladite image de programme non liée, produisant une image de programme entièrement liée dudit programme désigné avec des programmes de librairie attenants.

32. Système de mise en antémémoire d'image selon la revendication 31, dans lequel :

- ledit second moyen de liaison (32) agence de façon temporaire un quelconque desdits programmes de librairie non liés et stocke lesdits programmes de librairie agencés de façon temporaire dans ladite antémémoire de librairie ; et

- ledit second moyen de liaison (32) stocke ladite image de programme entièrement liée avec des programmes de librairie attenants avec ladite antémémoire d'image (144).

33. Système de mise en antémémoire d'image selon la revendication 32, dans lequel ledit programme désigné et lesdits programmes de librairie sont des programmes orientés objet.

34. Système de mise en antémémoire d'image selon la revendication 28, dans lequel ledit second moyen de recherche (26) effectuant ladite recherche en réponse à une demande de liaison dudit programme désigné lorsque ladite image de programme entièrement liée dudit programme désigné n'est pas trouvée par le système de liaison dynamique ; et

- un premier moyen de liaison (38) couplé audit second moyen de recherche pour lier ledit programme de librairie trouvé audit programme désigné dans le cas où ledit programme de librairie trouvé est trouvé dans ladite antémémoire de librairie.

35. Système de mise en antémémoire d'image selon la revendication 34, dans lequel ledit premier moyen de recherche comprend :

- un second moyen de recherche (26), couplé à ladite antémémoire de librairie (148), pour rechercher dans ladite antémémoire de librairie afin de déterminer si ladite antémémoire de librairie contient un programme de librairie non lié correspondant à un desdits programmes de librairie attenants.

36. Système de mise en antémémoire d'image selon la revendication 34, comprenant, de plus :

- un second moyen de recherche (26), couplé à ladite mémoire dudit ordinateur, pour trouver lesdits programmes de librairie non liés correspondant auxdits programmes de librairie attenants ; et

- un second moyen de liaison (32), couplé audit second moyen de recherche, pour lier lesdits programmes de librairie non liés l'un avec l'autre et avec lesdits programmes de librairie trouvés et avec ladite copie non liée dudit programme désigné, produisant une image de programme entièrement liée dudit programme désigné avec des programmes de librairie attenants.

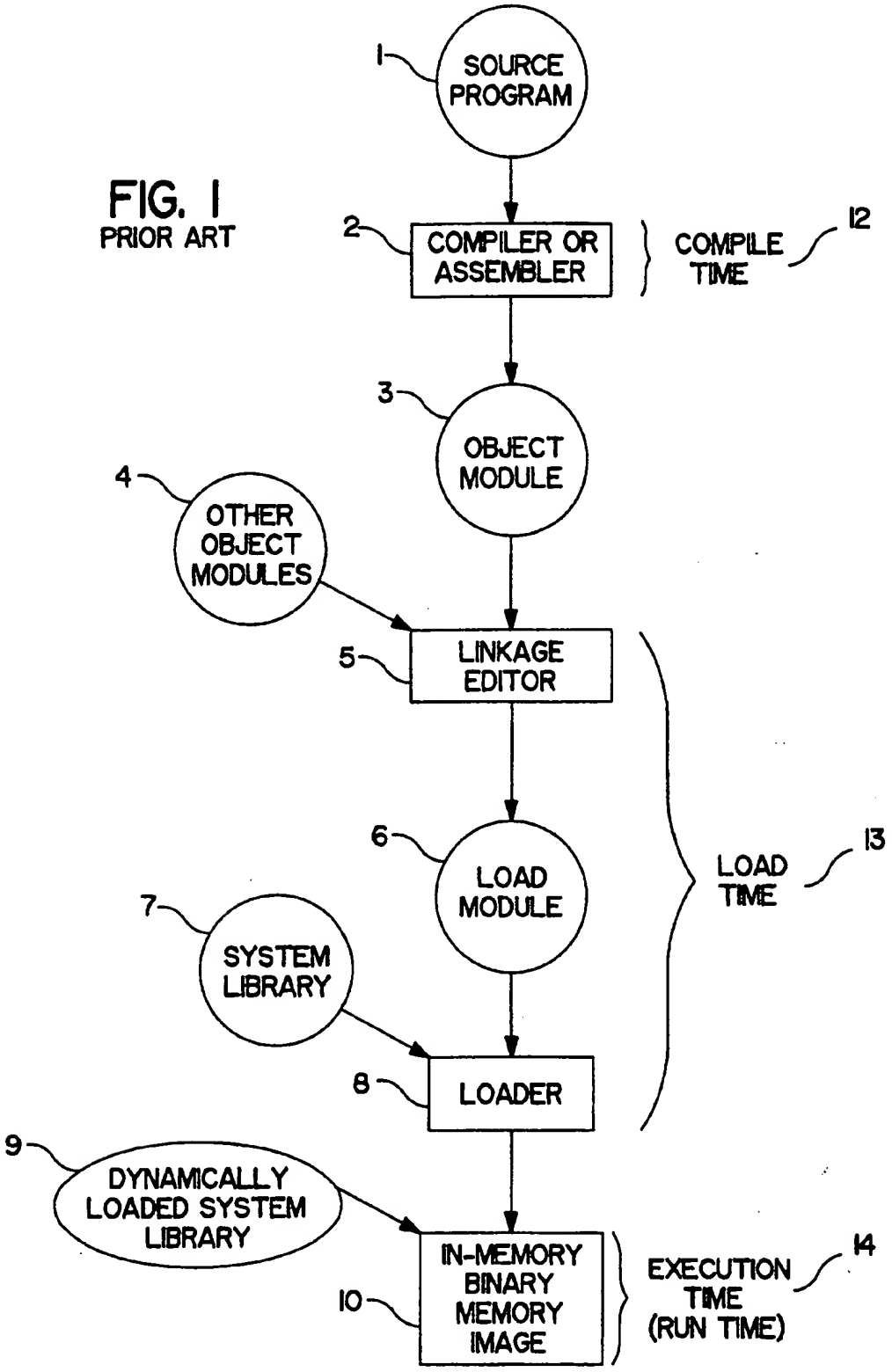
37. Système de mise en antémémoire d'image selon la revendication 36, dans lequel :

- ledit second moyen de liaison (32) agence, de façon temporaire, un quelconque desdits programmes de librairie non liés et stocke lesdits programmes de libraire agencés, de façon temporaire, dans ladite antémémoire de librairie.

38. Système de liaison dynamique selon la revendication 37, dans lequel lesdits programmes de librairie dans ladite antémémoire de librairie (148) sont partiellement liés auxdits programmes de librairie.

39. Système de liaison dynamique selon la revendication 38, dans lequel lesdits programmes d'application et lesdits programmes de librairie sont des programmes orientés objet.

FIG. 1
PRIOR ART



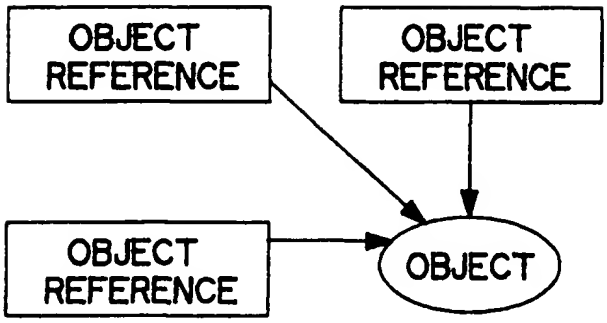


FIG. 2a
PRIOR ART

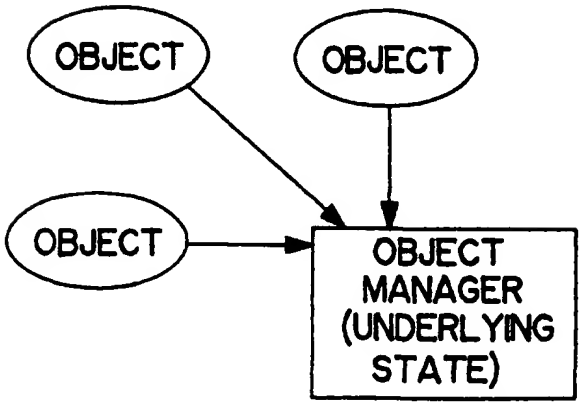


FIG. 2b

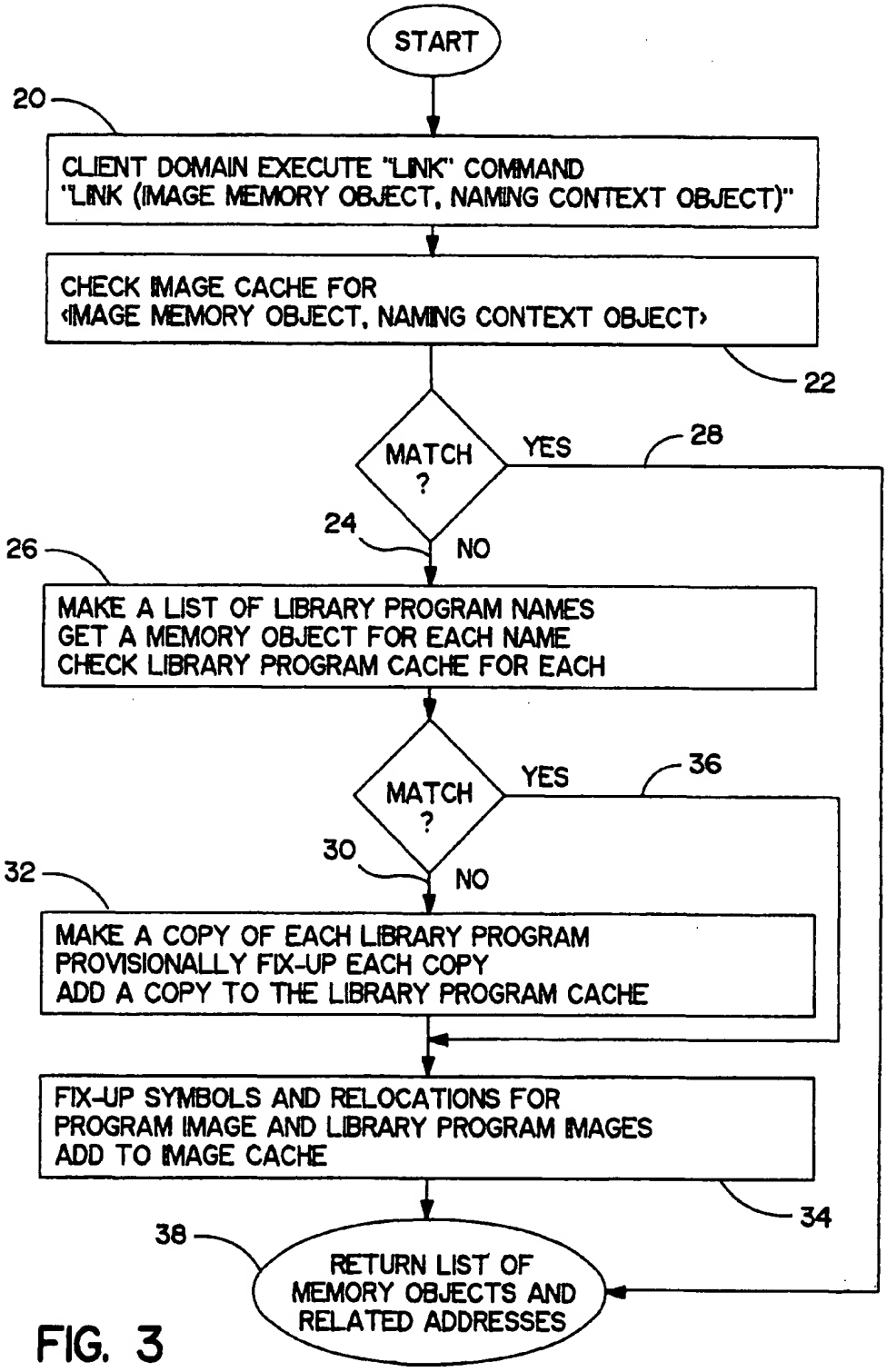

START
20
CLIENT DOMAIN EXECUTE "LINK" COMMAND
"LINK (IMAGE MEMORY OBJECT, NAMING CONTEXT OBJECT)"
CHECK IMAGE CACHE FOR
‹IMAGE MEMORY OBJECT, NAMING CONTEXT OBJECT›
22
MATCH ?
YES
28
24
NO
26
MAKE A LIST OF LIBRARY PROGRAM NAMES
GET A MEMORY OBJECT FOR EACH NAME
CHECK LIBRARY PROGRAM CACHE FOR EACH
MATCH ?
YES
36
30
NO
32
MAKE A COPY OF EACH LIBRARY PROGRAM
PROVISIONALLY FIX-UP EACH COPY
ADD A COPY TO THE LIBRARY PROGRAM CACHE
FIX-UP SYMBOLS AND RELOCATIONS FOR
PROGRAM IMAGE AND LIBRARY PROGRAM IMAGES
ADD TO IMAGE CACHE
34
38
RETURN LIST OF
MEMORY OBJECTS AND
RELATED ADDRESSES

FIG. 3

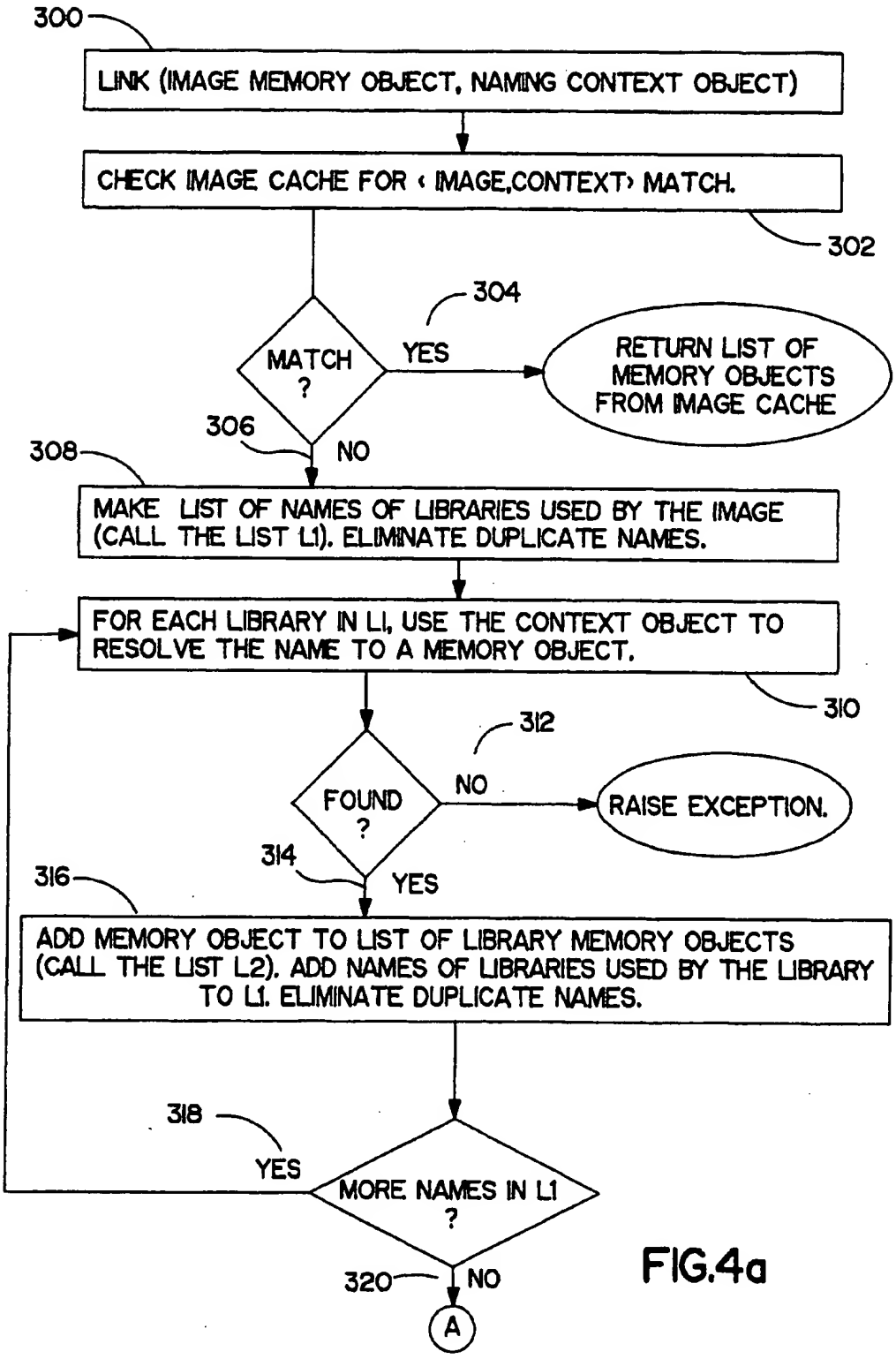

300
LINK (IMAGE MEMORY OBJECT, NAMING CONTEXT OBJECT)
CHECK IMAGE CACHE FOR ‹ IMAGE,CONTEXT› MATCH.
302
304
MATCH ?
YES
RETURN LIST OF MEMORY OBJECTS FROM IMAGE CACHE
306
NO
308
MAKE LIST OF NAMES OF LIBRARIES USED BY THE IMAGE (CALL THE LIST L1). ELIMINATE DUPLICATE NAMES.
FOR EACH LIBRARY IN L1, USE THE CONTEXT OBJECT TO RESOLVE THE NAME TO A MEMORY OBJECT.
310
312
FOUND ?
NO
RAISE EXCEPTION.
314
YES
316
ADD MEMORY OBJECT TO LIST OF LIBRARY MEMORY OBJECTS (CALL THE LIST L2). ADD NAMES OF LIBRARIES USED BY THE LIBRARY TO L1. ELIMINATE DUPLICATE NAMES.
318
YES
MORE NAMES IN L1 ?
320
NO
A

FIG.4a

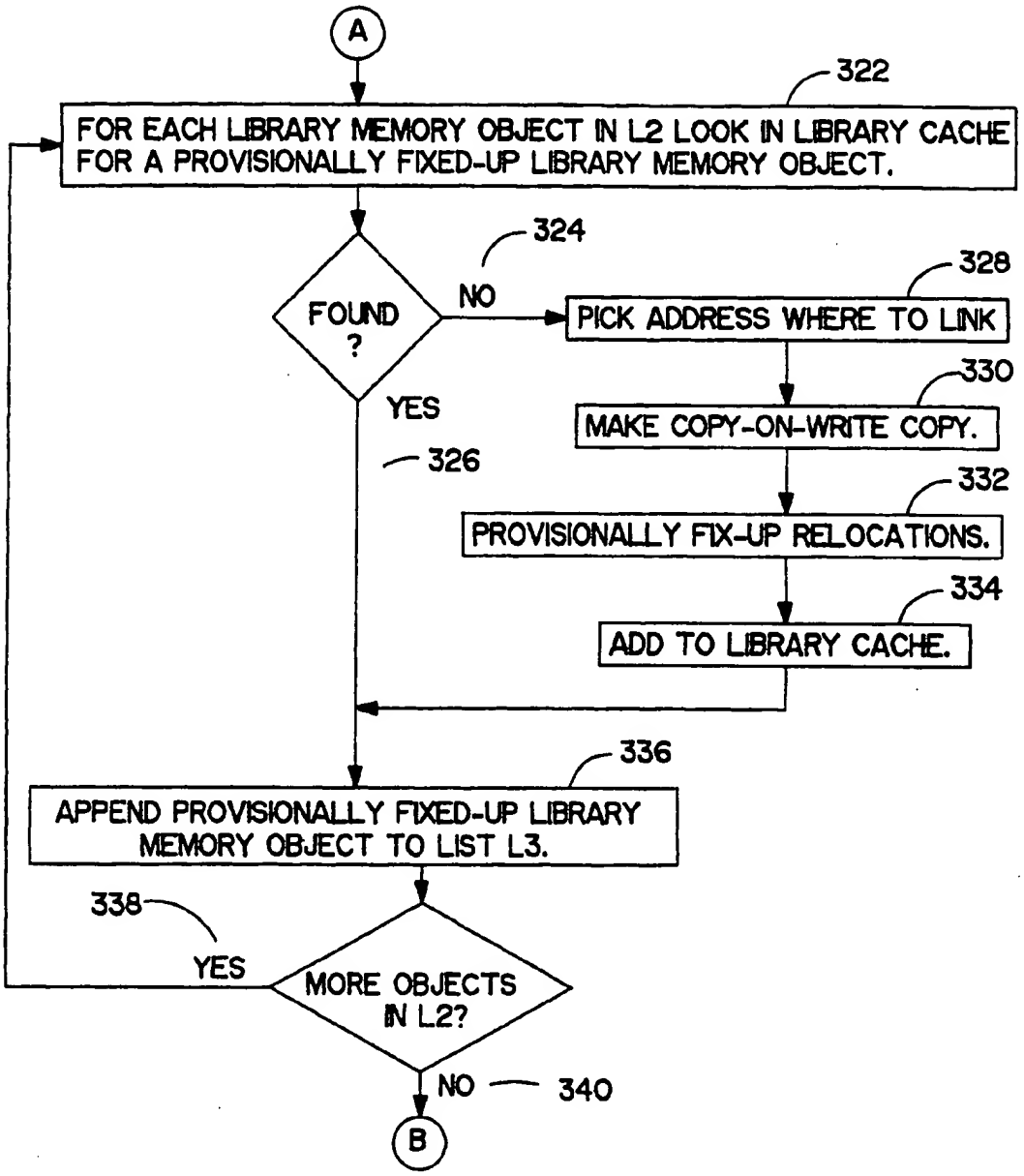

A
322
FOR EACH LIBRARY MEMORY OBJECT IN L2 LOOK IN LIBRARY CACHE FOR A PROVISIONALLY FIXED-UP LIBRARY MEMORY OBJECT.
324
FOUND ?
NO
328
PICK ADDRESS WHERE TO LINK
330
MAKE COPY-ON-WRITE COPY.
YES
326
332
PROVISIONALLY FIX-UP RELOCATIONS.
334
ADD TO LIBRARY CACHE.
336
APPEND PROVISIONALLY FIXED-UP LIBRARY MEMORY OBJECT TO LIST L3.
338
YES
MORE OBJECTS IN L2?
NO
340
B

FIG. 4b

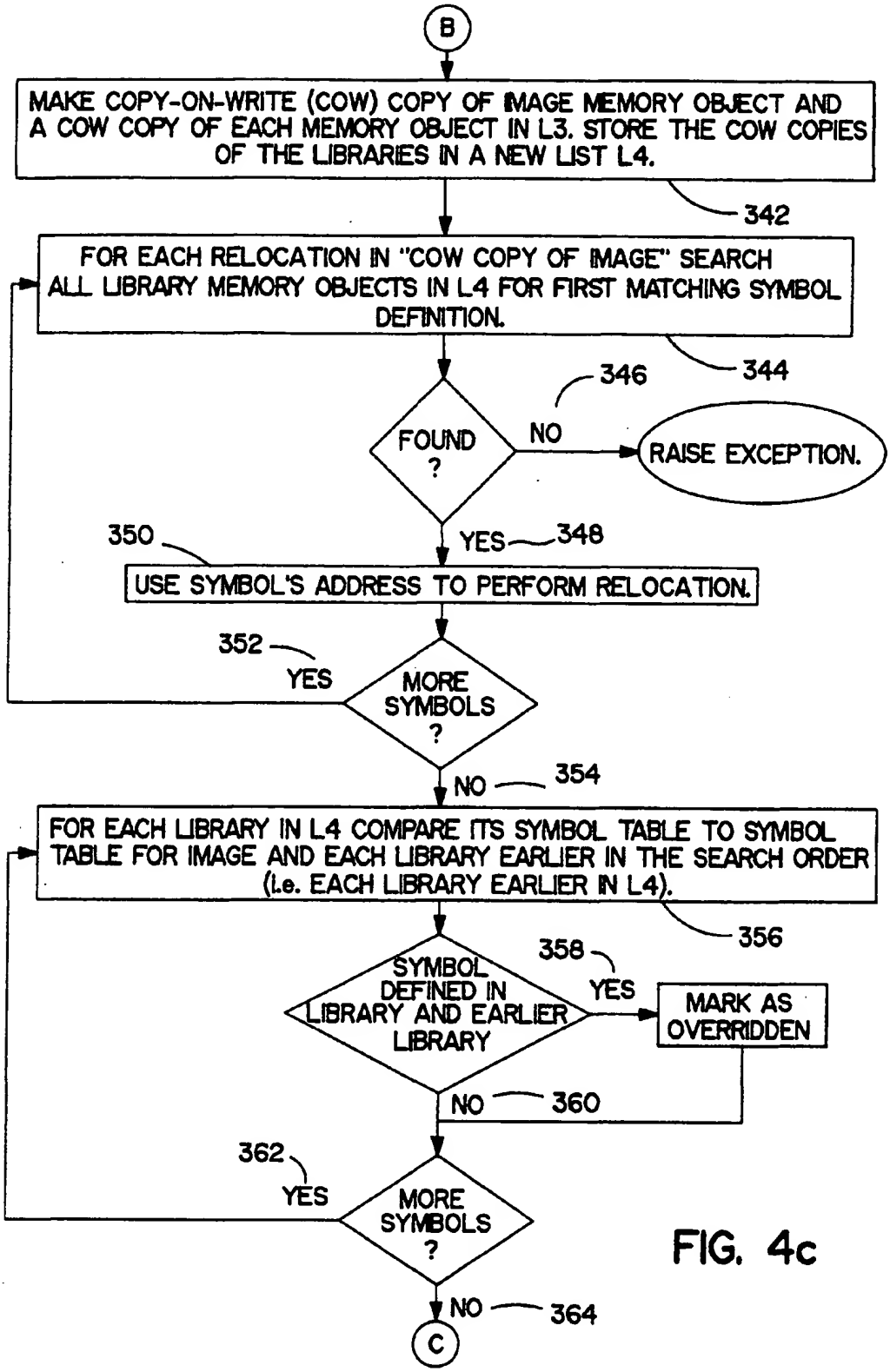

B
MAKE COPY-ON-WRITE (COW) COPY OF IMAGE MEMORY OBJECT AND A COW COPY OF EACH MEMORY OBJECT IN L3. STORE THE COW COPIES OF THE LIBRARIES IN A NEW LIST L4.
342
FOR EACH RELOCATION IN "COW COPY OF IMAGE" SEARCH ALL LIBRARY MEMORY OBJECTS IN L4 FOR FIRST MATCHING SYMBOL DEFINITION.
344
346
FOUND ?
NO
RAISE EXCEPTION.
350
YES
348
USE SYMBOL'S ADDRESS TO PERFORM RELOCATION.
352
YES
MORE SYMBOLS ?
NO
354
FOR EACH LIBRARY IN L4 COMPARE ITS SYMBOL TABLE TO SYMBOL TABLE FOR IMAGE AND EACH LIBRARY EARLIER IN THE SEARCH ORDER (i.e. EACH LIBRARY EARLIER IN L4).
356
358
SYMBOL DEFINED IN LIBRARY AND EARLIER LIBRARY
YES
MARK AS OVERRIDDEN
NO
360
362
YES
MORE SYMBOLS ?
NO
364
C

FIG. 4c

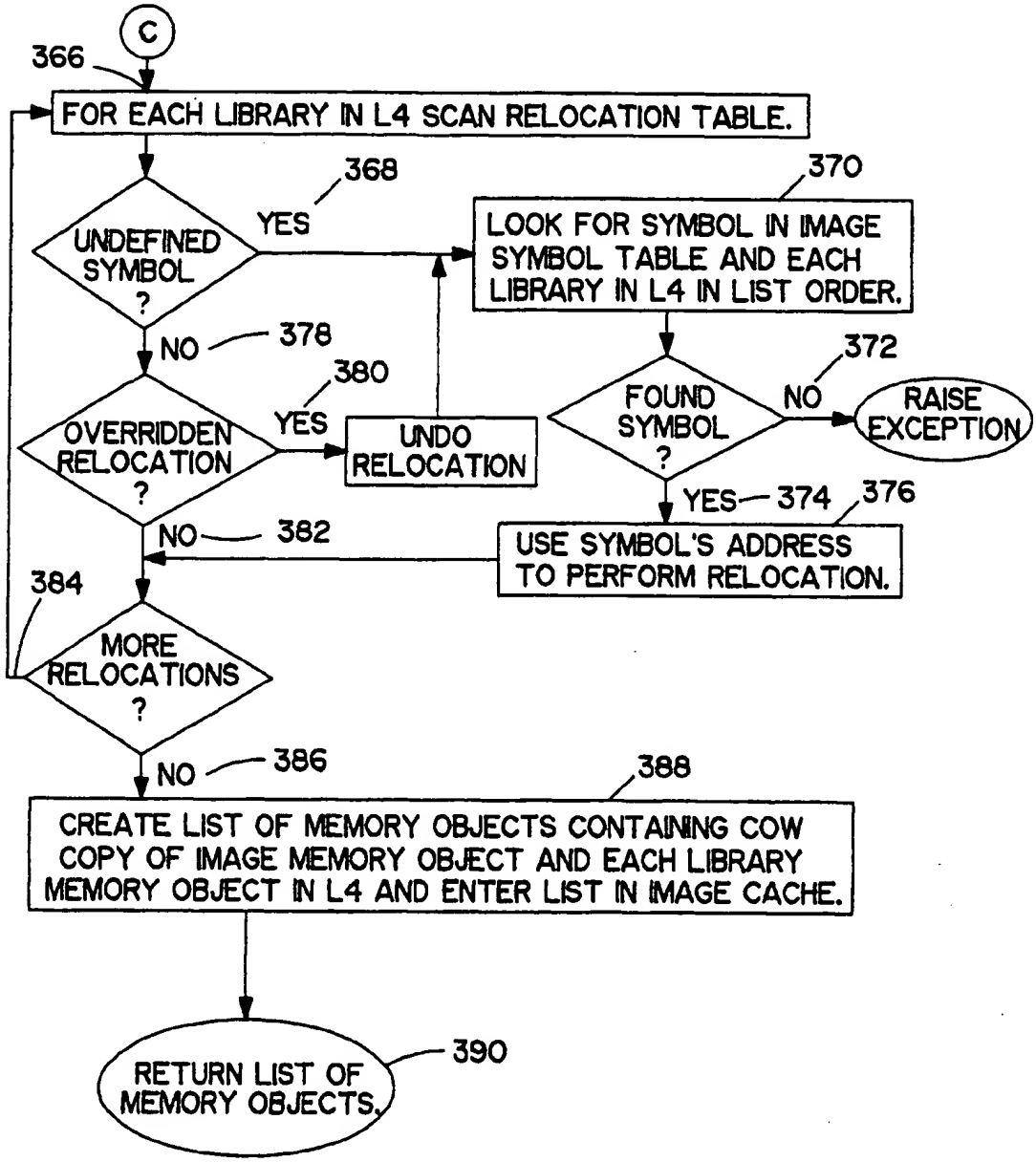

C
366
FOR EACH LIBRARY IN L4 SCAN RELOCATION TABLE.
368
UNDEFINED SYMBOL ?
YES
370
LOOK FOR SYMBOL IN IMAGE SYMBOL TABLE AND EACH LIBRARY IN L4 IN LIST ORDER.
NO
378
380
OVERRIDDEN RELOCATION ?
YES
UNDO RELOCATION
372
FOUND SYMBOL ?
NO
RAISE EXCEPTION
YES
374
376
USE SYMBOL'S ADDRESS TO PERFORM RELOCATION.
NO
382
384
MORE RELOCATIONS ?
NO
386
388
CREATE LIST OF MEMORY OBJECTS CONTAINING COW COPY OF IMAGE MEMORY OBJECT AND EACH LIBRARY MEMORY OBJECT IN L4 AND ENTER LIST IN IMAGE CACHE.
RETURN LIST OF MEMORY OBJECTS.
390

FIG. 4d

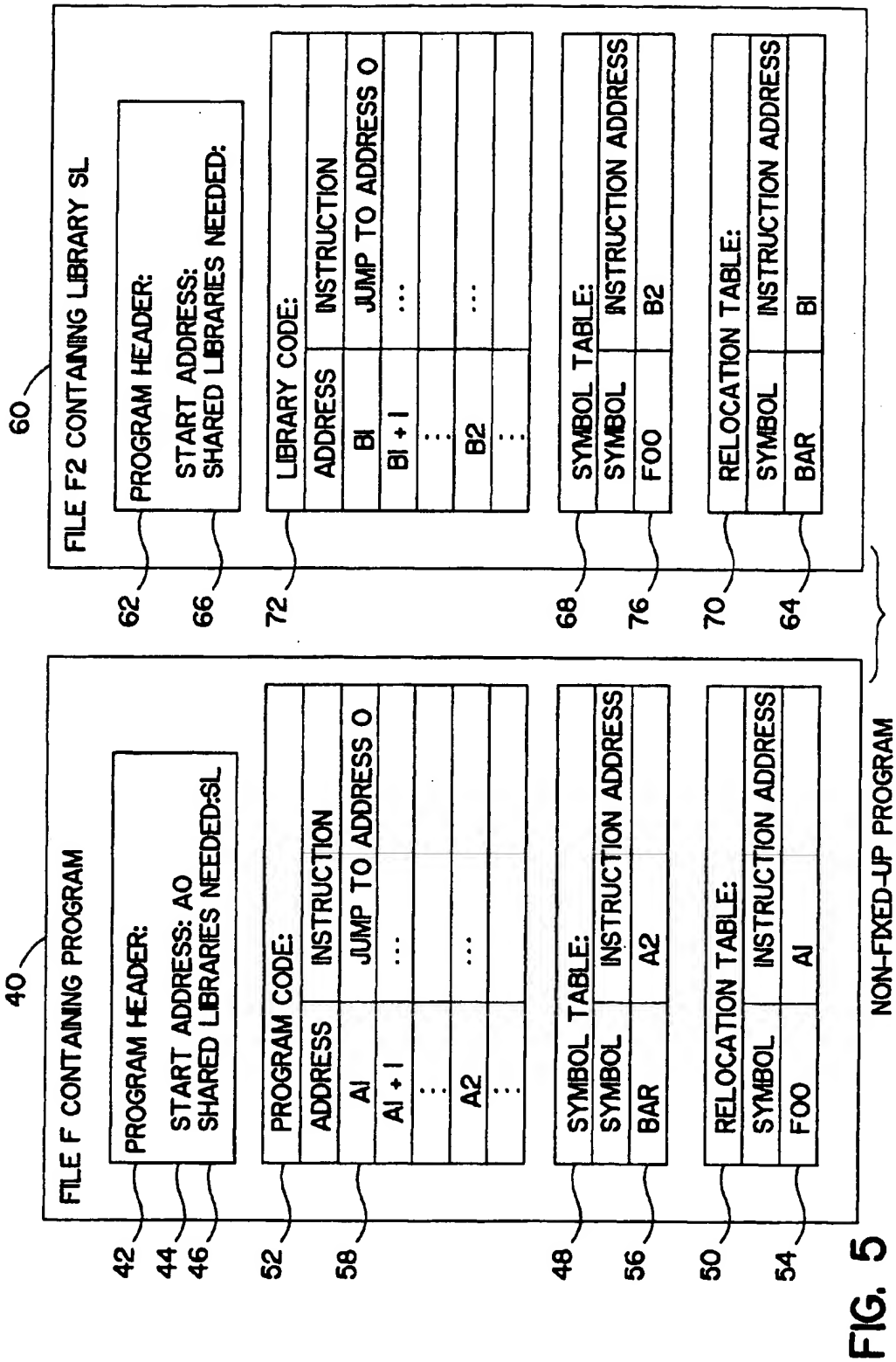

FILE F CONTAINING PROGRAM
40
PROGRAM HEADER:
START ADDRESS: AO
SHARED LIBRARIES NEEDED:SL
42
44
46
PROGRAM CODE:
ADDRESS
INSTRUCTION
AI
JUMP TO ADDRESS O
AI + I
A2
52
58
SYMBOL TABLE:
SYMBOL
INSTRUCTION ADDRESS
BAR
A2
48
56
RELOCATION TABLE:
SYMBOL
INSTRUCTION ADDRESS
FOO
AI
50
54
NON-FIXED-UP PROGRAM
FILE F2 CONTAINING LIBRARY SL
60
PROGRAM HEADER:
START ADDRESS:
SHARED LIBRARIES NEEDED:
62
66
LIBRARY CODE:
ADDRESS
INSTRUCTION
BI
JUMP TO ADDRESS O
BI + I
B2
72
SYMBOL TABLE:
SYMBOL
INSTRUCTION ADDRESS
FOO
B2
68
76
RELOCATION TABLE:
SYMBOL
INSTRUCTION ADDRESS
BAR
BI
70
64

FIG. 5

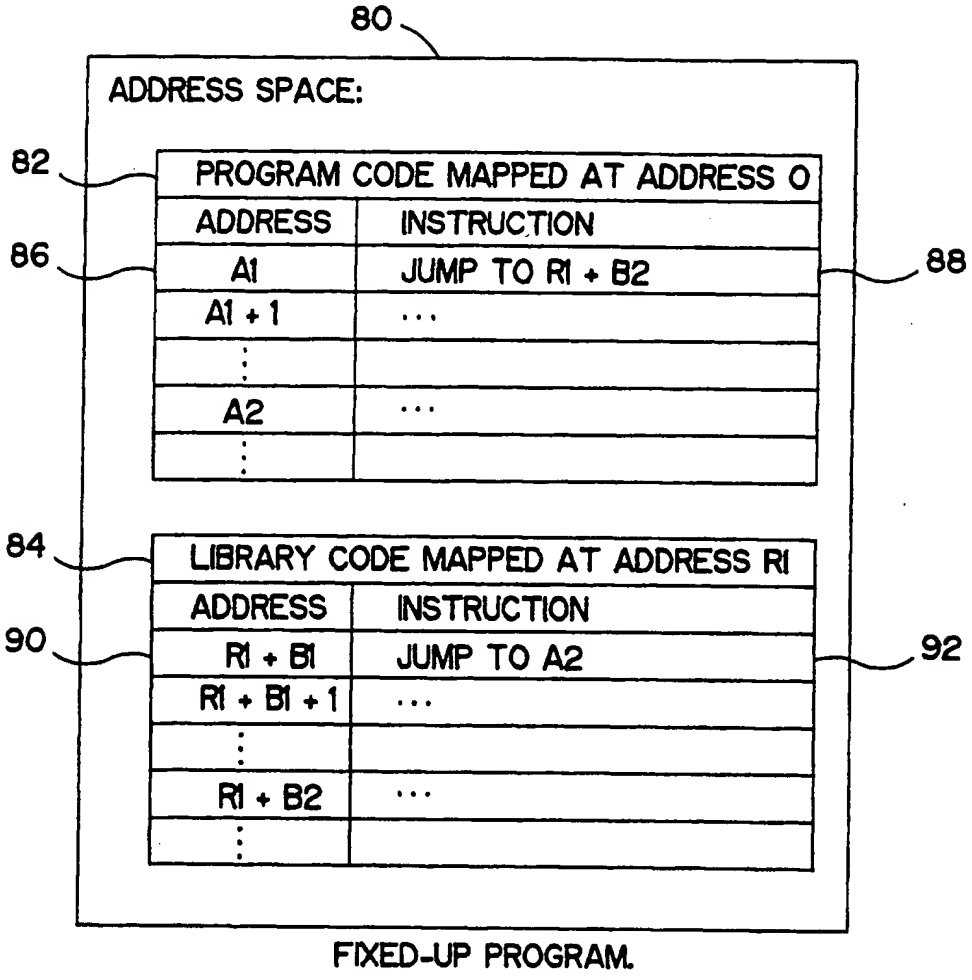
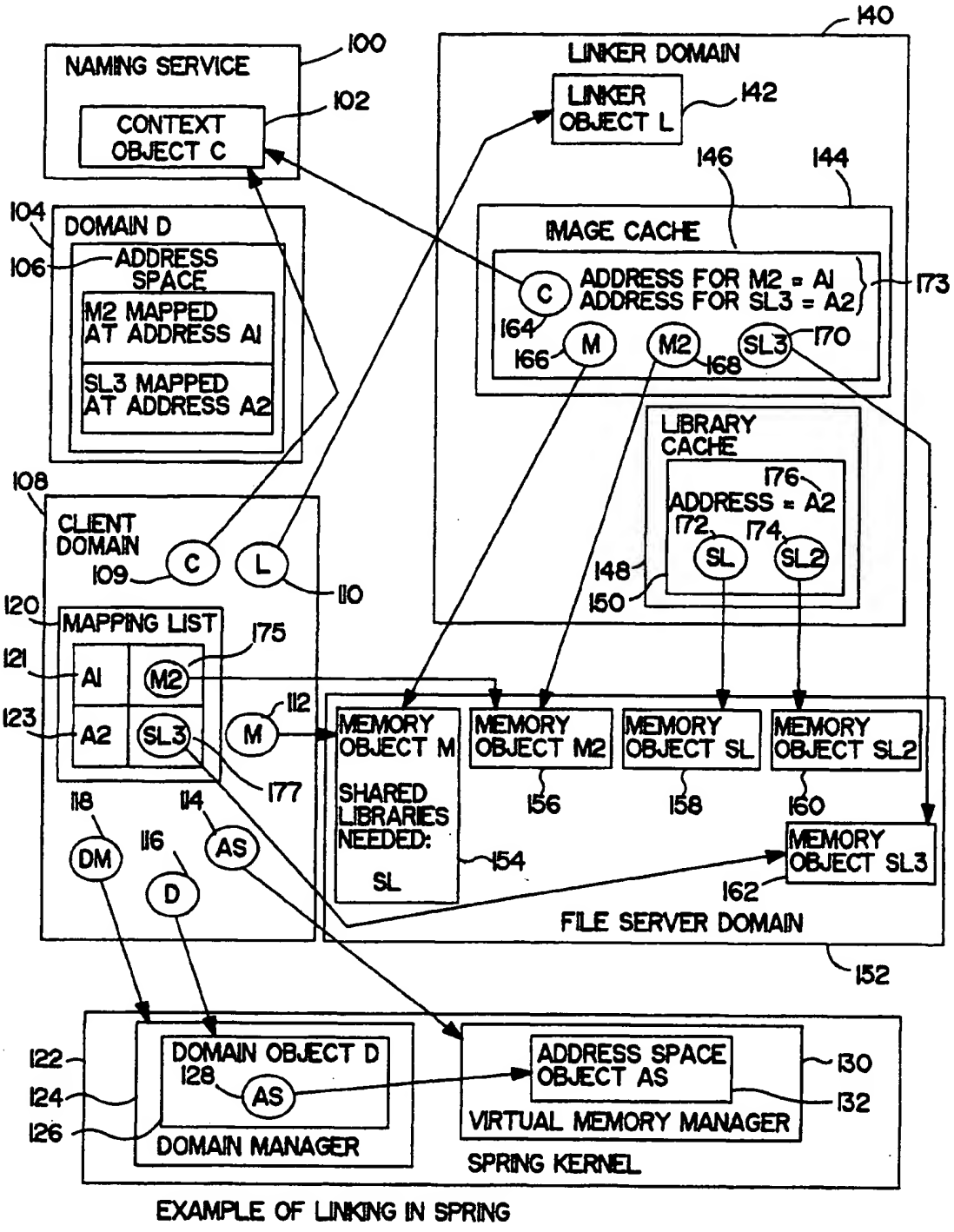

80
ADDRESS SPACE:
82
PROGRAM CODE MAPPED AT ADDRESS O
ADDRESS
INSTRUCTION
86
A1
JUMP TO R1 + B2
88
A1 + 1
...
A2
...
84
LIBRARY CODE MAPPED AT ADDRESS R1
ADDRESS
INSTRUCTION
90
R1 + B1
JUMP TO A2
92
R1 + B1 + 1
...
R1 + B2
...
FIXED-UP PROGRAM.

FIG. 6



EXAMPLE OF LINKING IN SPRING

FIG. 7